НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 «Програмна інженерія»

на тему: «Середовище розроблення програм на мові ASAMPL»

Виконав:
студент IV курсу, групи КП-52
Крисюк Андрій Михайлович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Сулема Є.С. _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н.,
Онай М.В. _____

Рецензент:
В.о. завідувача кафедри ММСА інституту прикладного системного аналізу, доцент, к.т.н.,
Тимощук О.Л. _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки – 6.050103 «Програмна інженерія»

ЗАТВЕРДЖУЮ
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2018 р.

ЗАВДАННЯ
на дипломний проект студенту

Крисюку Андрію Михайловичу

1. Тема проекту «Середовище розроблення програм на мові ASAMPL», керівник проекту Сулема Євгенія Станіславівна, к.т.н., доцент, затверджені наказом по університету від «22» травня 2019 р. № 1331-С.

2. Термін подання студентом проекту «14» червня 2019 р.

3. Вихідні дані до проекту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз існуючих рішень поставленої задачі;
   - обгрунтування вибору засобів реалізації;
   - розроблення алгоритмів та програмних модулів;
   - аналіз розробленого застосунку.

5. Перелік обов'язкового графічного матеріалу:
   - діаграма класів (креслення);
   - діаграма використання (креслення);
   - діаграма послідовності (креслення);
   - приклад роботи системи підсвітки синтаксису (плакат).

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2018 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Вивчення літератури за тематикою проекту	14.11.2018	
2.	Розроблення та узгодження технічного завдання	28.11.2018	
3.	Розроблення структури застосунку	15.12.2018	
4.	Підготовка матеріалів першого розділу дипломного проекту	19.12.2018	
5.	Розроблення дизайну графічного інтерфейсу	03.02.2019	
6.	Підготовка матеріалів другого розділу дипломного проекту	20.02.2019	
7.	Програмна реалізація застосунку	10.03.2019	
8.	Тестування застосунку	17.03.2019	
9.	Підготовка матеріалів третього розділу дипломного проекту	02.04.2019	
10.	Підготовка матеріалів четвертого розділу дипломного проекту	14.04.2019	
11.	Підготовка графічної частини дипломного проекту	22.04.2019	
12.	Оформлення документації дипломного проекту	27.05.2019	

Студент А.М. Крисюк

Керівник проекту Є.С. Сулема

АНОТАЦІЯ

Дана робота присвячена програмному застосунку – середовищу розроблення програм на мові ASAMPL.

У роботі виконано порівняльний аналіз існуючих інтегровних середовищ розробки, проаналізовано основні засоби та модулі, реалізовані в подібних рішеннях, обґрунтовано вибір технологій та допоміжних бібліотек для реалізації даного програмного застосунку. Розроблений застосунок надає можливість розробляти програми мовою ASAMPL, аналізує написаний код на наявність синтаксичних помилок, дозволяє відлагоджувати написані програми. Реалізоване середовище розроблення значно спрощує роботу з кодом за рахунок надання користувачу можливості редагувати код за допомогою повноцінного текстового редактору з функціями автоматичного доповнення коду, виділення кольором ключових слів мови, компіляції та запуску написаних програм.

У даному дипломному проекті розроблено: програмного застосунку з графічним інтерфейсом, модулі обробки коду, що реалізовують функції підсвітки синтаксису мови та автодоповнення коду, а також засоби відлагодження програм мовою ASAMPL.

ABSTRACT

This work is devoted to a software application – a program development environment in ASAMPL language.

In the work the comparative analysis of existing integrated development environments is made, the basic tools and modules implemented in such solutions are analyzed, the choice of technologies and auxiliary libraries for the implementation of this software application is grounded. The developed application provides the ability to develop programs in ASAMPL language, analyzes the written code for the presence of syntax errors, allows you to debug written programs. The implemented development environment greatly simplifies code work by providing the user with the ability to edit code using a full text editor with automatic code completion features, syntax highlighting, compilation, and start-up of written programs.

This graduation project has developed: a software application with a graphical interface, code processing modules that implement language syntax highlighting functions and code auto-alignment, as well as software debugging tools in ASAMPL language.

ДП.045440-01-90 Середовище розроблення програм на мові ASAMPL. Відомість проекту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проекту		
ДП.045440-02-91	Середовище розроблення	5	
	програм на мові		
	ASAMPL. Технічне		
	завдання		
ДП.045440-03-81	Середовище розроблення	57	
	програм на мові		
	ASAMPL. Пояснювальна		
	записка		
ДП.045440-04-51	Середовище розроблення	4	
	програм на мові		
	ASAMPL. Програма та		
	методика тестування		
ДП.045440-05-34	Середовище розроблення	9	
	програм на мові		
	ASAMPL. Керівництво		
	користувача		
ДП.045440-06-99	Середовище розроблення	1	
	програм на мові		
	ASAMPL. Архітектура		
	середовища розроблення.		
	Діаграма класів		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-07-99	Середовище розроблення	1	
	програм на мові		
	ASAMPL. Функціональні		
	можливості застосунку.		
	Діаграма використання		
ДП.045440-08-99	Середовище розроблення	1	
	програм на мові		
	ASAMPL. Процес		
	компіляції. Діаграма		
	послідовності		
ДП.045440-09-98	Середовище розроблення	1	
	програм на мові		
	ASAMPL. Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

СЕРЕДОВИЩЕ РОЗРОБЛЕННЯ ПРОГРАМ НА МОВІ ASAMPL

Технічне завдання

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проекту:

_____ Є.С. Сулема

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ А.М. Крисюк

2018

ЗМІСТ

1. Найменування та галузь застосування…….3
2. Підстава для розроблення…….....3
3. Призначення розробки……...3
4. Вимоги до програмного продукту……3
5. Вимоги до проектної документації……..4
6. Етапи проектування……...5
7. Порядок тестування розробки……..5

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Середовище розроблення програм на мові ASAMPL.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для спрощення та прискорення процесу розроблення програм на мові ASAMPL та забезпечення засобів їх відлагодження.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Застосунок має забезпечувати наступні функціональні можливості:

1) створення, відкриття та збереження файлів та проектів;
2) копіювання, вставка, вирізання, відміна та повтор останніх дій;
3) автодоповнення коду;
4) синтаксичний аналіз коду;
5) компіляція коду в середовищі розроблення;
6) відлагодження коду.

Розробку виконати мовами C# та C++, для реалізації графічного інтерфейсу використати бібліотеку Windows Forms.

Додаткові вимоги:

1) можливість зміни налаштуваннь середовища та редактору коду;
2) автокомпіляція коду;
3) виведення помилок в лог компіляції;
4) можливість редагування налаштувань системи за допомогоюконфігураційних файлів;
5) дизайн сторінок з використанням в якості базових білого та синьо-зеленого кольорів.

5. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
    - «Архітектура середовища розроблення. Діаграма класів»;
    - «Функціональні можливості застосунку. Діаграма використання»;
    - «Процес компіляції. Діаграма послідовності».

6. ЕТАПИ ПРОЕКТУВАННЯ

Вивчення літератури за тематикою роботи..... 14.11.2018

Розроблення та узгодження технічного завдання..... 28.11.2018

Розроблення структури застосунку..... 15.12.2018

Розроблення дизайну графічного інтерфейсу..... 03.02.2019

Програмна реалізація застосунку..... 10.03.2019

Тестування застосунку 17.03.2019

Підготовка матеріалів текстової частини проекту 14.04.2019

Підготовка матеріалів графічної частини проекту..... 22.04.2019

Оформлення технічної документації проекту..... 27.05.2019

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"
Науковий керівник кафедри
_____ І. А. Дичка
"___" _____ 2019 р.

СЕРЕДОВИЩЕ РОЗРОБЛЕННЯ ПРОГРАМ НА МОВІ ASAMPL
Пояснювальна записка

ДП.045480-03-81

"ПОГОДЖЕНО"

Керівник проекту:

_____ Є. С. Сулема

Нормоконтроль:

_____ М. В. Онай

Виконавець:

_____ А.М.Крисюк

2019

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ 3
ВСТУП 5
1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ПОСТАВЛЕНОЇ ЗАДАЧІ 6
1.1. Поточний стан розвитку інтегрованих середовищ розроблення 6
1.2. Середовища розроблення 7
1.3. Обґрунтування актуальності розробки 11
1.4. Висновки до розділу 1 12
2. ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ 13
2.1. Вибір мови програмування 13
2.2. Вибір засобів реалізації графічного інтерфейсу. 19
2.3. Вибір сторонніх бібліотек 21
2.4. Висновки до розділу 2 22
3. РОЗРОБЛЕННЯ АЛГОРИТМІВ ТА ПРОГРАМНИХ МОДУЛІВ 23
3.1. Архітектура середовища розроблення 23
3.2. Особливості реалізації засобів автодоповнення та синтаксичного аналізу коду 30
3.3. Особливості реалізації відлагоджувача 34
3.4. Висновки до розділу 3 38
4. АНАЛІЗ РОЗРОБЛЕНОГО ЗАСТОСУНКУ 39
4.1. Практичне застосування розробленого застосунку 39
4.2. Реалізовані функціональні можливості 39
4.3. Напрямки подальшого вдосконалення 47
4.4. Висновки до розділу 4 50
ВИСНОВКИ 51
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ 52
ДОДАТКИ 57

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

IDE – Інтегроване середовище розроблення (англ. Integrated development environment), комплексне програмне рішення для розроблення програмного забезпечення. Зазвичай складається з редактору початкового коду, інструментів автоматизації складання та відлагодження програм.

Дебаґер – комп'ютерна програма, яка використовується для тестування і виправлення вад інших програм.

IntelliSense – технологія автодоповнення коду, розроблена компанією Microsoft.

Плаґін – *plug-in* – додаток, незалежно скомпільований програмний модуль, що динамічно підключається до основної програми, призначений для розширення або використання її можливостей. Належить до загального програмного класу додатків. Плаґіни, зазвичай, виконуються у вигляді динамічних бібліотек.

Фреймворк – *framework* – комплекс програмних рішень, що складається з допоміжних програм, бібліотек коду та API. Використання фреймворків спрощує створення програмних продуктів, надаючи стандартний підхід до їх побудови та розгортання.

RCP – набір модулів для розроблення програмного забезпечення, надає програмістам засоби побудови власних застосунків на основі готових модулів.

API – *прикладний програмний інтерфейс* – це набір визначень взаємодії різнотипного програмного забезпечення. Є абстракцією, що описує надану програмним компонентом функціональність.

Java Community Process (JCP) – сформований у 1998 році формальний процес, який дозволяє зацікавленим особам брати участь у формуванні майбутніх версій специфікацій платформ мови Java.

DirectX – це набір API функцій, розроблених для простого і ефективного вирішення завдань, пов'язаних з ігровим і відеопрограмуванням під Microsoft Windows.

Common Language Runtime (CLR) – це компонент пакету Microsoft .NET Framework, віртуальна машина, на якій виконуються всі мови платформи .NET Framework.

XAML (*eXtensible Application Markup Language*) – декларативна мова розмітки. З точки зору моделі програмування .NET Framework мова XAML спрощує створення користувацького інтерфейсу для програми .NET Framework.

Graphics Device Interface, GDI – один з трьох основних компонентів, що складають користувацький інтерфейс Microsoft Windows.

Model-View-ViewModel – це шаблон проектування, що застосовується під час проектування архітектури застосунків (додатків). MVVM полегшує відокремлення розроблення графічного інтерфейсу від розроблення бізнес логіки (бек-енд логіки), відомої як модель (можна також сказати, що це відокремлення представлення від моделі).

Windows Presentation Foundation (WPF) – система для побудови клієнтських застосунків Windows з візуально привабливими можливостями взаємодії з користувачем, графічна (презентаційна) підсистема у складі .NET Framework.

Лог – журнал або спеціальний файл, у якому накопичується зібрана службова та статистична інформація про події в системі (програмі).

ВСТУП

В сучасному світі важко уявити розробку програмних застосунків будь-якою мовою без використання середовища розроблення (IDE). Використання IDE суттєво спрощує процес та скорочує час написання коду, що є корисним як для IT-компаній, так і для розробників проектів із відкритим вихідним кодом.

Розроблення та використання IDE стала можливою лише із появою перших терміналів або консолей, які можливо було використати безпосередньо для написання коду. Із розвитком програмних систем ускладнювалися й інструменти їх розроблення. Сьогодні IDE представляє собою масштабні програмні рішення, деколи кроссплатформенні, що включають в себе багато функціональних можливостей для роботи з кодом, таких як автодоповнення коду, автоматизація складання проектів, відлагодження програм та навіть системи контролю версій. Існують безліч IDE пристосованих до роботи з однією чи декількома мовами, і щодня список цих програмних засобів можна поповнювати новими назвами. З постійно зростаючого ринку середовищ розроблення програмного забезпечення можна зробити висновок, що задача створення подібного застосунку є актуальною, особливо для нових мов програмування, які ще не мають власного середовища, як, наприклад, мова програмування обробки мультимодальних даних ASAMPL [1].

Даний дипломний проект присвячено розробці середовища розроблення програм на мові ASAMPL та його основних компонентів, аналогічних існуючим рішенням для інших мов програмування.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ПОСТАВЛЕНОЇ ЗАДАЧІ

1.1. Поточний стан розвитку інтегрованих середовищ розроблення

Зі збільшенням кількості коду в програмних рішеннях зростають і вимоги до IDE, за допомогою яких вони створюються. Окрім досить тривіальних вимог до стабільності роботи програмного застосунку, які зустрічаються в кожному проекті, та більш спеціалізованих вимог до швидкості роботи та виконання задач, розроблення інтегрованих середовищ також має досить багато вимог до полегшення процесу роботи з кодом.

Сучасні IDE пропонують широкий спектр можливостей, які можуть суттєво спростити роботу над кодом.

Перш за все, більшість середовищ розроблення та текстових редакторів коду мають можливості автодоповнення та синтаксичного аналізу коду. Автодоповнення сприяє швидшому написанню коду, надаючи можливість закінчити строку або слово коду будь-якою із наявних у програмі чи підключених бібліотеках функцією, змінною чи назвою файлу. Однією із можливостей системи автодоповнення може бути використання сніппетів – шаблонів широковживаних синтаксичних конструкцій (циклів, умовних конструкцій, тощо).

Синтаксичний аналіз коду надає ряд інших можливостей. В першу чергу, ще на етапі написання коду синтаксичний аналізатор вказує розробнику на помилки у коді, а більш складні аналізатори також можуть вказувати на синтаксичні конструкції, що можуть не мати сенсу чи можуть працювати нестабільно. Також однією з основних функцій синтаксичного аналізу коду є спрощення орієнтування у коді за допомогою візуальних знаків, наприклад, підсвічення різними кольорами назв функцій, змінних, ключових слів мови, арифметичних виразів.

Іншою важливою частиною сучасної IDE є інструменти автоматизації складання проекту. Ці інструменти дозволяють перетворити файли із вихідним кодом у один виконуваний бінарний файл, за допомогою

компілятора або інтерпретатора мови програмування. Наявність таких інструментів у IDE виключає можливість помилки програміста при зв'язуванні різних файлів із вихідним кодом у один проект; прискорює процес компіляції та компонування; а як результат - зменшує витрати часу та грошей, поліпшує якість продукту.

Однією із найбільших проблем при написанні коду є його відлагодження, тобто пошук помилок та дефектів у програмному забезпеченні. Для спрощення цього процесу більшість сучасних IDE надають спеціальний інструмент для відлагодження коду – дебаґер. Найчастіше зустрічаються наступні функціональні можливості дебаґера: покрокове виконання коду програми та відстеження значення змінних у пам’яті. Наявність дебаґера є важливим фактором використання тієї чи іншої IDE, а деколи і мови програмування, навіть якщо вона більше підходить для вирішення певної задачі.

1.2. Середовища розроблення

Кожного дня з’являються оновлення існуючих чи принципово нові середовища розроблення, але найбільшим попитом користуються вже перевірені часом рішення, як, наприклад, Microsoft Visual Studio, Intellij IDEA та Eclipse.

1.2.1. *Visual Studio*

Visual Studio – одна з найпопулярніших серій середовищ розроблення, створена компанією Microsoft, що дозволяє розроблювати веб-сайти та веб-застосунки, консольні програми та програми із графічним інтерфейсом, для усіх платформ, що підтримуються Microsoft.

Основою Visual Studio є редактор вихідного коду, що підтримує технологію IntelliSense та найпростіші засоби рефакторингу коду. IntelliSense – це набір функцій, що надає відомості про код безпосередньо в редакторі, а в деяких випадках автоматично генерує невеликі уривки коду.

Фактично, технологія представляє собою базову документацію, що вбудована в редактор. Функції IntelliSense залежать від мови, що використовується.

Серед інструментів цього середовища розроблення є:

- вбудований відлагоджувач;
- редактор форм графічних інтерфейсів;
- редактор веб-сторінок;
- дизайнер класів;

Одним із важливих інструментів для веб-програмування є вбудований веб-сервер, який дозволяє запускати веб-сайт безпосередньо з середовища програмування, а також підвищує безпеку, виключаючи ймовірність отримання доступу до тестового серверу із зовнішніх пристроїв, оскільки тестовий сервер може мати з'єднання лише з локальним комп'ютером.

Окрім наявної великої кількості вбудованих засобів для роботи з кодом, Visual Studio підтримує підключення сторонніх плагінів що дозволяє значно розширити функціональність будь-яких засобів, починаючи від технології IntelliSense та закінчуючи доданням системи контролю версій.

Основним недоліком Visual Studio є її «важкість» [12] - через велику кількість функціональних можливостей відкриття, завантаження та запуск проектів може потребувати значних ресурсів комп’ютера.

Поточні версії Visual Studio включають підтримку наступних компонентів:

- Visual Basic .NET;
- Visual C++;
- Visual C#;
- Visual F#.

Оскільки якісних IDE для мов C#, Visual Basic та F# довго не існувало, Visual Studio швидко здобула популярність, а серед спільноти розробників C++ вона є чи не найпопулярнішим середовищем розроблення [5].

Завдяки просуванню компанією Microsoft, підтримці багатьох розробників програмного забезпечення, великій кількості функціональних можливостей для роботи з платформою .Net та іншими мовами, Visual Studio впевнено лідирує у всіх сучасних рейтингах середовищ розроблення.

1.2.2. *Eclipse*

Eclipse – це інтегроване середовище розроблення програмного забезпечення, розроблене та підтримуване Eclipse Foundation, куди входять такі постачальники програмного забезпечення як IBM, Oracle, Borland. Від початку Eclipse був розроблений для роботи із мовою програмування Java, але з розвитком та створенням нових плагінів став також підтримувати C, C++, COBOL, ADA, Fortran, PHP, Perl, Python, R, Ruby, Scala та ін.

Eclipse є програмним забезпеченням з відкритим кодом, що дозволяє розробникам власноруч створювати нові плагіни та конфігурувати IDE під власні потреби.

Eclipse запрограмовано мовою Java, тому він є платформонезалежним, що робить його також популярним серед розробників, що користуються Linux, Solaris та Mac OS X. Для створення елементів користувацького інтерфейсу використовується платформозалежна бібліотека SWT – Standart Widget Toolkit.

Іншою особливістю Eclipse [13] є використання підходу RCP (Rich Client Platform), що дозволяє використовувати його для створення майже будь-якого клієнтського програмного забезпечення.

Як і Visual Studio, Eclipse підтримує систему контролю версій, інструменти автоматичного складання проектів, та засоби відлагодження коду для мови Java, та для інших мов програмування із встановленням відповідних плагінів. Eclipse є фреймворком для розроблення інших інструментів, а також надає великий набір API для створення модулів. IDE розроблені на платформі Eclipse використовуються для створення програмного забезпечення на різних мовах програмування, оскільки Eclipse

є платформою для розроблення будь-яких інтегрованих середовищ програмування і розширень для себе, за принципом "Плагіни для Eclipse розробляються в самій Eclipse".

Архітектурно Eclipse можна розділити на два елементи:

- Виконуюче середовище – Eclipse Runtime, в якому виконуються коди розширень та модулів. Воно забезпечує базові функції програми: управління розширеннями та оновленнями, взаємодія з операційною системою, забезпечення роботи довідкової системи.
- Середовище розроблення – відповідає за управління основними елементами програми, їх розташуванням і налаштуваннями, управління проектами, відлагодження та автоматичну збірку проектів, пошук у файлах та командну розробку.

Враховуючи безкоштовне поширення цієї IDE, вона стала корпоративним стандартом для багатьох компаній-розробників на мові Java. Згідно міжнародних маркетингових досліджень Eclipse у своїй роботі використовують більше 2 млн розробників.

1.2.3. *Intellij IDEA*

Intellij IDEA – інтегроване середовище розроблення, створене компанією JetBrains. Швидко набула популярність як перше середовище розроблення для Java з широким набором інтегрованих інструментів для рефакторингу.

На сьогодні система представлена у вигляді двох версій – вільної Community Edition та комерційної Ultimate Edition.

Community версія підтримує велику кількість інструментів, серед яких є засоби проведення тестування (TestNG та JUnit), системи контролю версій (Mercurial, Git, CVS), засоби автоматичного складання проектів (Maven, Ant, Gradle), а також велику кількість мов програмування: Java, Scala, Go, Kotlin, Python, Lua та інші. Ще однією особливістю Intellij IDEA є можливість розроблення мобільних застосунків для Android. Також Community версія

надає графічний редактор для проектування користувацьких інтерфейсів з використанням бібліотеки Swing для мови Java. Серед інших інструментів наявних у безкоштовній версії варто виділити наступні: редактор регулярних виразів, система перевірки коректності коду, системи контролю за виконанням завдань і доповнення для імпорту та експорту проектів з Eclipse.

Ultimate версія має розширені функціональні можливості, зокрема вона підтримує більший набір мов програмування (наприклад, PHP, Ruby, Python, JavaScript, CoffeeScript та інші), а також велику кількість технологій, таких як Google App Engine, Java EE, Grails, Hibernate, Play Framework та інші). Додатково Ultimate версія надає редактор UML-діаграм та систему підрахунку покриття коду.

Серед недоліків системи часто називають [11] необхідність часових витрат на освоєння, оскільки в Intellij IDEA існує велика кількість поєднань гарячих клавіш, які необхідно запам'ятовувати. Також деякі користувачі скаржаться на незграбний інтерфейс.

Завдяки великій кількості функціональних можливостей, інструментів для розроблення мобільних застосунків та корпоративних технологій для різних платформ, Intellij IDEA займає місце в десятці кращих інтегрованих середовищ розроблення.

1.3. Обґрунтування актуальності розробки

Основною метою середовищ розроблення є спрощення процесу створення програмного забезпечення тією чи іншою мовою. В цей процес входить як написання коду, так і його читання, тому наявність в IDE таких функцій як аналіз синтаксису, форматування та рефакторинг коду, автодоповнення значно пришвидшують роботу розробника, що є корисним для корпоративної та індивідуальної розробки. Інколи навіть існування якісного середовища розроблення впливає на вибір розробником конкретної

мови програмування. А отже наявність такого редактору може стати одним із факторів популяризації мови.

Аналізуючи ринок середовищ розроблення програмного забезпечення, можна зробити висновок, що серед розробників є попит на якісні програмні застосунки, націлені на спрощення написання коду. Враховуючи відсутність подібного програмного застосунку для мови ASAMPL, буде доцільним створити інтегроване середовище розроблення програмного забезпечення для цієї мови, яке буде реалізувати наступний типовий інструментарій [22]:

- редактор коду;
- засоби автодоповнення;
- засоби синтаксичного аналізу;
- засоби відлагодження програм.

Вказані функціональні можливості мають спростити та прискорити процес розроблення програмного забезпечення мовою ASAMPL.

1.4. Висновки до розділу 1

В даному розділі проведено аналіз поточного стану розвитку середовищ розроблення програмного забезпечення. На прикладі Visual Studio, Intellij IDEA, Eclipse IDE було розглянуто основні функціональні можливості, необхідні для реалізації в даному дипломному проекті.

На основі аналізу ринку можна стверджувати, що необхідність у якісних програмних застосунках такого типу існує постійно. У зв’язку з відсутністю такого застосунку для мови ASAMPL є актуальним створення інтегрованого середовища розроблення програмного забезпечення.

2. ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Вибір мови програмування

Не дивлячись на широкий вибір мов програмування, наявний сьогодні, для реалізації інтегрованих середовищ розроблення великі компанії та невеликі команди проектів з відкритим вихідним кодом використовують вже перевірені часом мови. Аналізуючи існуючі на ринку рішення, можна виокремити наступних лідерів у цій сфері - Java, C++, C#.

Продукт буде розділено на дві окремі частини у зв’язку з особливостями їх реалізації. Першою частиною є редактор коду для розроблення якого необхідна мова з наступними ознаками:

- Простота синтаксису.
- Наявність чітко визначених стандартів. Відповідність стандартам спрощує підтримку проекту в майбутньому, дозволяє використовувати перевірені та стандартизовані засоби.
- Наявність вичерпної документації та великої спільноти розробників, сторонніх якісних бібліотек.
- Наявність потужної бібліотеки для розроблення графічних інтерфейсів, та редактору для зручного їх створення.

Другою частиною є відлагоджувач для розроблення якого необхідна мова з наступними особливостями:

- статична типізація;
- можливість управління пам’яттю.

2.1.1. *Мова програмування Java*

Java – високорівнева інтерпретована об’єктно-орієнтована мова програмування.

Однією з основних особливостей, закладених розробниками, стала кросплатформність мови, реалізована за допомогою віртуальної машини

Java, яка транслює програми в байт-код, який може виконуватись на будь-якій платформі, для якої існує віртуальна машина.

Завдяки віртуальній машині існує й інша особливість Java – гнучка система безпеки. Виконання програми на мові Java контролюється віртуальною машиною, і у випадку перевищення повноважень (наприклад, несанкціонованого доступу до даних) викликається системне переривання.

Специфікації Java формуються за допомогою Java Community Process (JCP) – процесу, що дозволяє розробникам вносити власні побажання щодо формування майбутніх специфікацій платформ Java. JCP базується на Java Specification Request (JSR) – формальних документах, що описують зміни, які пропонується внести до наступної версії мови. Прийняті JSR виносяться на розгляд спільноти розробників та експертів, і приймаються лише після обговорення запропонованих змін.

Оскільки Java є однією з найпопулярніших мов програмування у світі, спільнота розробників постійно виносить на обговорення багато питань з приводу реалізації різних засобів. У зв’язку з цим, існує велика кількість сторонніх бібліотек та фреймфорків для вирішення будь-яких задач.

Компанія Oracle, яка є фактичним розробником мови Java на даний момент, надає вичерпну документацію для всіх стандартних бібліотек та пакетів Java. Окрім офіційної документації, існує велика кількість літератури присвяченої цій мові та використанню її у різних сферах.

Java є мовою із статичною типізацією та автоматичним управлінням пам'яттю, а її синтаксис дуже схожий із мовами C/C++. Але порівнюючи з програмами написаними на C++, програми написані на Java виконуються в середньому у півтора-два рази довше та споживають значно більшу кількість пам’яті.

Одними з найпопулярніших засобів розроблення графічних інтерфейсів мовою Java є наступні бібліотеки:

- Swing – графічна бібліотека [16], що відзначилася вичерпною офіційною документацією, гнучкістю компонентів та досить

низьким порогом входження. Чимало сучасних IDE включають графічні редактори для Swing-форм.

- SWT, Standart Widget Toolkit – бібліотека, що використовує компоненти операційної системи для створення форм, що дозволяє розробляти кросплатформні застосунки. Перекладає задачу звільнення ресурсів на розробника, що збільшує вірогідність витоку пам'яті. У порівнянні з Swing є більш складною для освоєння та використання.
- JavaFX – бібліотека [21], реалізована на основі графічного конвеєра, що значно пришвидшує роботу застосунку. Має широкий набір компонентів, зовнішній вигляд яких можливо редагувати за допомогою CSS-стилів.

2.1.2. *Мова програмування C#*

C# – об'єктно-орієнтована мова програмування зі статичною типізацією.

Під час розроблення C# команда спеціалістів Microsoft спиралася на вже існуючі рішення такі як C++, Java, Modula. Завдяки цьому вдалося уникнути деяких проблематичних рішень, як, наприклад, множинне успадкування у C++. C# базується на Common Language Runtime (CLR – загальномовне виконуюче середовище) технології, тобто як і у Java, код C# спочатку транслюється у проміжний байт-код, що дозволяє писати застосунки для різних платформ, не змінюючи код самої програми.

Компанія Microsoft розробляє стандарти мови C# для найновіших її версій (як 7.0), але міжнародними організаціями із розроблення стандартів специфіковано лише версії 5.0 (ECMA у 2017 році) та 2.0 (ISO/IEC у 2006 році) [36].

Враховуючи популярність мови, кількість літератури та інтернет-ресурсів для вивчення є дуже великою. Офіційна документація, представлена на сайті Microsoft, надає багато прикладів використання

стандартної бібліотеки. Також присутня велика кількість сторонніх бібліотек розроблених спільнотою. За даними офіційного сайту сховища пакетів NuGet, на сьогодні існує більше 148 тис. пакунків.

Синтаксис C# є C-подібним, тому є дуже схожим на C++ та Java. Також C# реалізовує статичну типізацію. Керування пам'яттю залишає на автоматичний збирач сміття реалізований в CLR. У порівнянні з C++ швидкодія C# менша: для різних задач, способів їх реалізації та вимірювання швидкодії час виконання програми може бути від 10 до 80 відсотків нижчим за час виконання аналогічної програми на C++.

Microsoft пропонує наступні бібліотеки для розроблення графічних інтерфейсів на платформі .Net:

- WPF, Windows Presentation Forms – платформа, основою якої є графічна технологія DirectX, що дозволяє використовувати апаратне прискорення для збільшення швидкодії. Для роботи із графічними елементами існує вбудований в Visual Studio редактор, а також мова розмітки XAML.
- Windows Forms – API, що базується на технології GDI/GDI+. Має великий набір інструментів для розроблення користувацьких інтерфейсів та вичерпну документацію, що робить дану бібліотеку досить простою для використання.

2.1.3. *Мова програмування C++*

C++ – компільована статично типізована мова програмування.

Стандартизацією мови C++ займається комітет до складу якого входять спеціалісти з міжнародної організації стандартизації (ISO) та міжнародної електротехнічної комісії (IEC). Комітет оновлює стандарти C++ майже кожні 3-4 роки.

Відповідно до принципів проектування мови C++, викладених Страуструпом у книзі «Дизайн та еволюція C++» [2], C++ розроблювався аби підтримувати можливість використовувати багато стилів

програмування, бути максимально платформонезалежним та забезпечувати якомога вищу швидкодію.

Однією з основних особливостей C++ є можливість ручного керування пам’яттю. Дана особливість є великим плюсом при розробці низькорівневих застосунків та тих, що потребують максимальної швидкодії. Але недоліком цього є велика кількість помилок під час розроблення, а як наслідок збільшення часових та грошових витрат на проект.

Оскільки C++ є досить старою і все ще популярною мовою програмування, у мережі знаходиться велика кількість матеріалів. Вичерпну документацію надає Microsoft для свого компілятора Visual C++. Велика спільнота розробників постійно доповнює мову новими сторонніми бібліотеками, найякіснішою з яких вважається бібліотека Boost.

Для розроблення графічних інтерфейсів на C++ було створено велику кількість графічних бібліотек, серед яких можна виділити наступні:

- Windows Forms – для Visual C++.
- Qt – кросплатформний фреймворк, що містить великий набір інструментів для розроблення та проектування користувацьких інтерфейсів. Командою-розробником представлена вичерпна документація, яка доповнюється великою кількістю матеріалів в мережі. Графічний редактор представлений у середовищі розроблення Qt Creator.
- wxWidgets – кросплатформна бібліотека з відкритим вихідним кодом. Основне застосування даної бібліотеки – саме розроблення графічних інтерфейсів, але також в ній наявна велика кількість інших функцій та інструментів. Проект має досить повну документацію, але деякі статті є застарілими. Також жодне із середовищ розроблення не надає вбудованих графічних редакторів для цієї бібліотеки.

2.1.4. *Порівняльна оцінка*

У табл.1 наведено порівняння розглянутих мов за наявністю особливостей встановлених вимогами проекту. Також наведені оцінки за п'ятибальною шкалою, для критеріїв які можливо оцінити таким чином. Оцінювання проводилось з врахуванням відповідності вимогам проекту та власного досвіду використання, а також на основі експертних оцінок колег, що працюють в провідних IT-компаніях.

Таблиця 1

Порівняння мов програмування

	C++	C#	Java
Наявність особливостей			
Стандарти	Так	Так	Так
Статична типізація	Так	Так	Так
Можливість управління пам'яттю	Так	Ні	Ні
Експертна оцінка			
Простота синтаксису	4	5	5
Вичерпність довідкової документації	4	5	4
Зручність використання графічних бібліотек	5	5	4
Зручність використання редакторів графічних інтерфейсів	5	5	4
Середня оцінка	4,5	5	4,25

Використання Java та C# у проектах дозволяє суттєво зменшити час розроблення прототипу продукту, оскільки ці мови є більш простими у використанні, порівняно з C++.

Платформа .Net надає можливість розробляти графічні інтерфейси мовами C++ та C#, а також поєднувати проекти, написані цими мовами, що є зручною перевагою.

Для реалізації текстового редактору буде використано мову C#, оскільки вона є простою у використанні та має широкий набір засобів для реалізації графічних інтерфейсів.

У якості мови для реалізації відлагоджувача буде використано C++, оскільки він задовольняє вимогу ручного управління пам’яттю, що є важливою складовою розроблення цієї частини проекту.

2.2. Вибір засобів реалізації графічного інтерфейсу

Платформа .Net пропонує для реалізації користувацьких інтерфейсів наступні засоби:

- Windows Forms;
- WPF.

Розглянемо дані технології детальніше.

2.2.1. *Windows Forms*

Windows Forms – частина Microsoft .Net Framework, що відповідає за графічний інтерфейс користувача. Windows Forms розглядається як заміна застарілої бібліотеки MFC, але не реалізує парадигми MVC. Для її використання потрібно встановити сторонні бібліотеки, однією з яких є, власне, розроблена Microsoft – User Interface Process Application Block.

Застосунок Windows Forms є подійно-орієнтованим, тобто його виконання визначається діями користувача чи системи. Недоліком такої архітектури є неможливість обробки декількох подій.

Microsoft надає вичерпну документацію для Windows Forms, а середовище розроблення Visual Studio надає інструменти для редагування форм.

2.2.2. *WPF*

WPF – підсистема Microsoft .Net Framework, що реалізована на основі векторної системи візуалізації, яка не залежить від роздільної здатності пристроїв виведення.

Однією з особливостей WPF є використання мови розмітки XAML, що базується на XML. Використання XAML дозволяє розділити роботу над застосунком для розробника і дизайнера.

Основною ідеєю використання WPF є відокремлення бізнес логіки застосунку від його графічного представлення, за допомогою таких шаблонів проектування, як Model-View-ViewModel. Модель в цьому шаблоні являє собою дані, необхідні для роботи застосунку. Модель вигляду здійснює зв’язок між моделлю даних та графічним інтерфейсом, надаючи для останнього інтерфейс для роботи з даними.

Microsoft активно просуває технологію WPF та надає вичерпну документацію до неї, а зростаюча популярність серед розробників призвела до появи великої кількості матеріалів.

2.2.4. *Порівняльна оцінка*

У порівнянні з WPF, Winforms вважається застарілою технологію. На даний момент Winforms все ще підтримуються Microsoft, але компанія заявила, що не буде розвивати їх надалі.

WPF має чимало переваг у зручності редагування елементів інтерфейсу та швидкодії [23], але тільки за умови використання шаблонів проектування на кшталт MVVM. В іншому випадку, швидкодія таких застосунків не має суттєвої переваги над аналогічним застосунком Winforms. Використання MVVM також робить WPF більш складним для освоєння.

У свою чергу, технологія Winforms простіша у використанні [14], і є достатньо функціональною у тих випадках, коли застосунок не потребує значної роботи з дизайном, складними графічними елементами та анімаціями.

Тому для реалізації графічного інтерфейсу проекту буде використано Windows Forms.

2.3. Вибір сторонніх бібліотек

Для реалізації текстового редактору буде використано бібліотеку ICSharpCode.TextEditor [3], яка надає велику кількість засобів для роботи з текстом.

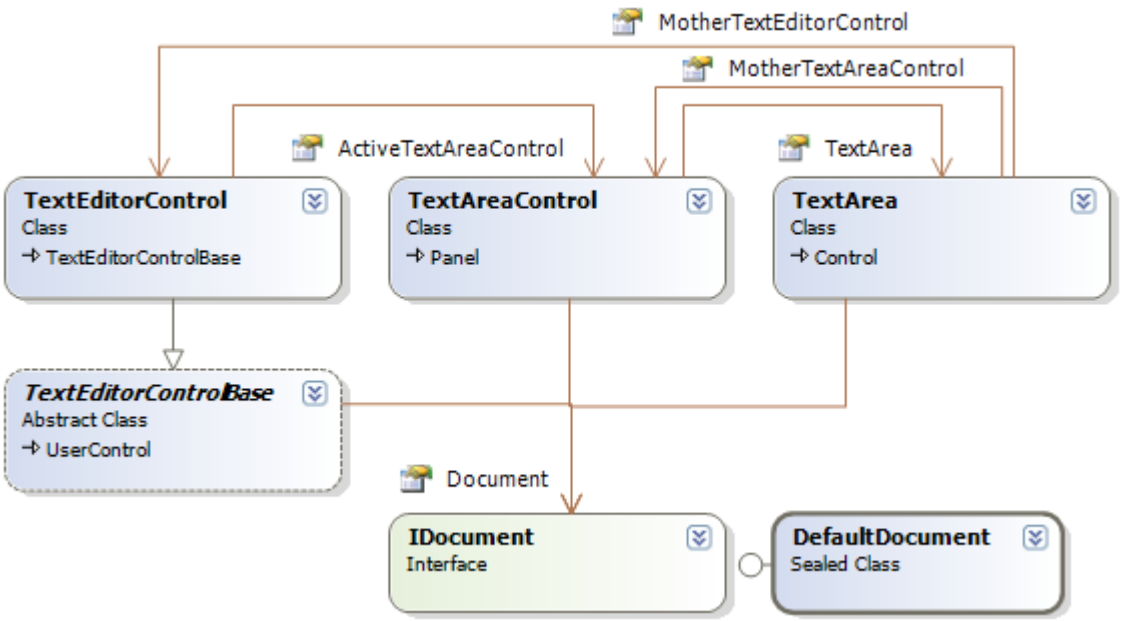


Рис. 1. Діаграма базових класів бібліотеки ICSharpCode.TextEditor

Інтерфейс *IDocument* [8], реалізований в класі *DefaulDocument*, надає чимало функціональних можливостей притаманних текстовому редактору:

- Анулювання чи повернення змін в тексті (undo/redo);
- Автоматична вставка табуляцій;
- Система підсвічення синтаксису;
- Пошук та заміна тексту;

- Управління текстовим буфером.

Не зважаючи на неповну документацію, розробники використовують дану бібліотеку в проектах з відритим вихідним кодом, що надає чимало прикладів використання необхідних функцій. З огляду на те, що бібліотека є досить потужним інструментом для роботи з текстовими елементами – є доцільним використати її при розробленні дипломного проекту.

2.4. Висновки до розділу 2

В рамках цього розділу дипломної роботи було проаналізовано доступні засоби розроблення проекту.

Були висунуті вимог до мови програмування:

- наявність вичерпної документації та додаткових ресурсів;
- наявність чітко визначених стандартів;
- наявність редакторів графічних інтерфейсів у середовищах розроблення;
- наявність статичної типізації та ручного управління пам’яті.

На основі даних вимог для реалізації проекту було обрано дві мови програмування: C# для текстового редактору та графічних інтерфейсів та C++ для реалізації відлагоджувача.

Для створення графічного інтерфейсу користувача було обрано технологію Windows Forms, яка є частиною платформи .Net.

Для реалізації текстового редактору також було обрано бібліотеку ICSharpCode.TextEditor.

3. РОЗРОБЛЕННЯ АЛГОРИТМІВ ТА ПРОГРАМНИХ МОДУЛІВ

3.1. Архітектура середовища розроблення

Інтерфейс середовища розроблення на мові ASAMPL розроблено у відповідності до існуючих аналогів, тому він має більшість елементів властивих подібним рішенням.

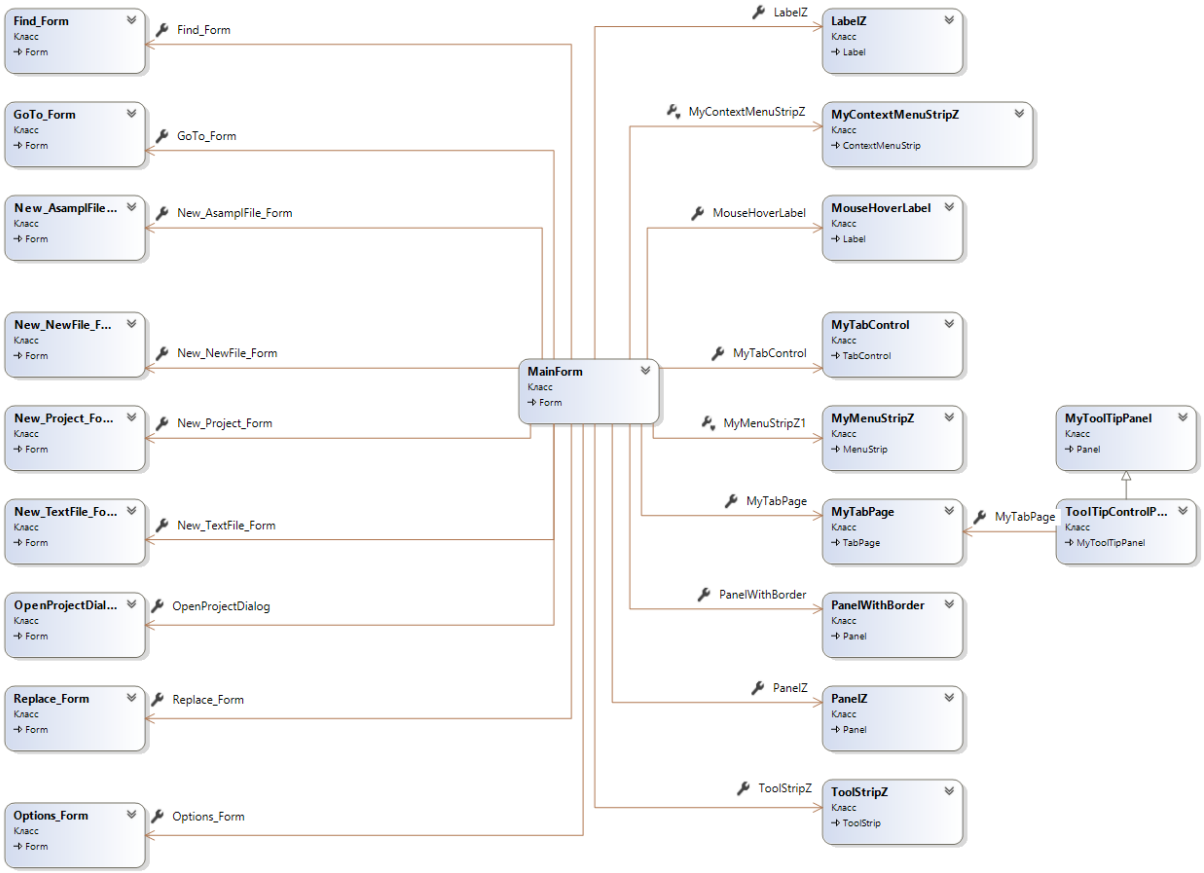


Рис. 2. Діаграма класів застосунку

На рис. 2 зображена діаграма класів, що реалізовані в застосунку. У його правій частині розташовані елементи форм, що застосовуються в головній формі застосунку, а в лівій частині розташовані форми, що викликаються при відповідних діях користувача.

Клас *MainForm* описує головну форму користувацького інтерфейсу та взаємодію з його елементами.

Основними елементами форми є:

- панель меню;
- панель інструментів;
- дерево файлів проекту;
- редактор;
- лог помилок.

Панель меню вміщує п'ять розділів меню, що згруповані за змістом представлених ними функцій.

Меню *File* надає доступ до функцій роботи з файлами та проектами, таких як створення нових файлів та проектів, збереження та закриття файлів, пошук останнього відкритого проекту для продовження роботи з ним. На даний момент пункт *New* цього меню надає можливість створити новий проект, текстовий файл, файл коду ASAMPL та файл із довільним користувацьким розширенням. При створенні нового проекту у файл проекту записується інформація про назву, розташування проекту на диску, при створенні файлів – шляхи до них у файловій системі. Файл проекту представляє собою файл у форматі XML із спеціальним розширенням *.asamplproj*.

При створенні нового файлу чи проекту викликається відповідні форми реалізовані в класах *New_NewFile_Form*, *New_TextFile_Form*, *New_AsamplFile_Form* та *New_Project_Form*. Перші три форми мають лише одне поле введення – для назви файлу, а також включають методи перевірки коректності назви файлу та обробники подій, що пов'язані з елементами форми. Кожна з форм також має метод для створення нового файлу, який перевіряє чи існує вже такий файл. Якщо такого файлу не існує у вказаній директорії проекту, метод створює нову вкладку в редакторі та маркує її, як не збережену, для подальшої обробки в головній формі. В обробнику події створення нового файлу в головній формі відбувається відкриття потоку для запису у створений файл та запис даних про його розташування до файлу

проекту. Також новий файл додається до дерева проекту у відповідній панелі.

Форма створення нового проекту відрізняється наявністю додаткового поля для введення шляху розташування проекту в файловій системі комп'ютера та реалізацією метода створення проекту. Після проходження перевірок на коректність даних та існування проекту, створюється XML-документ, до якого вносяться дані про розташування директорії та файлу проекту, його назву та назву середовища розроблення.

Меню *Edit* надає доступ до типових функцій роботи з редактором тексту. Окрім стандартних функцій копіювання та вставки, також реалізовані функції відміни останньої дії, повтору останньої дії, пошуку та заміни в тексті, перехід до обраної строки коду, вставка коментарів та навігація між документами, як за допомогою елементів меню, так і за допомогою гарячих клавіш.

Лістинг 1. Функція відміни останньої дії

```
private void Edit_UndoMenuItem_Click(object sender, EventArgs e){
    if (myTabControl.TabCount > 0)
    {
        if (myTabControl.SelectedTab.Text != "Start Page")
        {
            int select_index = myTabControl.SelectedIndex;
            var texteditor = (TextEditorControl)myTabControl.
                            TabPages[select_index].Controls[0];
            texteditor.Undo();
        }
    }
}
```

Обробники події відміни та повтору останньої дії мають схожу реалізацію. При виклику відбувається перевірка на існування вкладок з файлами відмінних від стартової сторінки, обирається поточна вкладка, і за допомогою приведення до класу *TextEditorControl* реалізованого в бібліотеці *ICSharpCode.TextEditor* викликається метод цього класу *Undo* чи *Redo*, в залежності від події.

При виклику обробника події пошуку в тексті створюється екземпляр класу *FindForm*. При отриманні даних для пошуку, відбувається перебір текстового контейнеру поточної вкладки. У випадку існування пошукового запиту в тексті, знайдені строки зберігаються в тимчасовому контейнері. При цьому у формі відображається повідомлення із номером строки коду, в якому було знайдено запит, а також фокус текстового редактора переноситься на відповідну строку. Для навігації по результатам пошуку використовується кнопка *FindNext*, обробник якої збільшує лічильник результатів на одиницю та відповідно переносить фокус текстового редактору на наступну строку з результатом.

Обробник події пошуку та заміни в тексті створює екземпляр класу *ReplaceForm*. Отримавши дані від користувача, викликається метод стандартного класу *RichTextBox,* що використовується для представлення тексту в редакторі:

```
rtb.Text = rtb.Text.Replace(findtext, replacetext);
```

Меню *View* надає користувачу можливість керування елементами інтерфейсу, такими як:

- розташування панелі вкладок – нижнє чи верхнє;
- наявність панелі статусу;
- наявність панелі інструментів;
- використання повноекранного режиму;
- управління кольоровою схемою середовища розроблення;
- наявність панелі з нумерацією строк;
- наявність дерева файлів проекту;
- наявність логу компіляції;
- виділення пробілів, табуляцій та символів закінчення рядку.

Меню *Run* надає користувачу доступ до налаштувань середовища розроблення, а також функцій компіляції та запуску коду.

Обробник події компіляції проекту передає до методу *CompileAsampl* назву поточного відкритого файлу з кодом та шлях до компілятора,

записаний в налаштуваннях середовища розроблення. Метод *CompileAsampl* викликає метод *Compile*, що відповідає за створення нового процесу компіляції.

Якщо процес компілятора було успішно запущено, метод *CompileAsampl* починає обробку повідомлення, отриманого від компілятора. У випадку знаходження компілятором помилки в коді, до логу компіляції вносяться дані про першу знайдену помилку, що є особливістю поточної версії компілятора мови ASAMPL. Після отримання виводу з помилками компілятора в методі відбувається пошук строк з наявними помилками, і фокус текстового редактора переноситься на строку коду вказану в логу компіляції.

Також, якщо відповідне налаштування вибране, при наявності помилок з'являється вікно зі списком помилок, що були отримані від компілятора.

Лістинг 2. Реалізація методу *Compile*

```
public bool Compile(String EXE, String WorkingDirectory,
                                                String FileName) {
      bool processStarted = false;

      if (File.Exists(EXE))
      {
          process.StartInfo.FileName = EXE;
          process.StartInfo.Arguments = FileName;
          process.StartInfo.WorkingDirectory = WorkingDirectory;
          process.StartInfo.CreateNoWindow = false;
          process.StartInfo.ErrorDialog = false;
          process.StartInfo.UseShellExecute = false;
          process.StartInfo.RedirectStandardOutput = true;
          process.StartInfo.RedirectStandardError = true;
          processStarted = process.Start();
      }
      else
          MessageBox.Show("Unable to compile ASAMPL file. Check your
                  ASAMPL Path settings: Current ASAMPL Path : ");
      return processStarted;}
```

Обробник події доступу до налаштувань створює екземпляр класу *OptionsForm*. Форма налаштувань має дві вкладки – *General* та *Editor*. Перша містить налаштування кольорової теми середовища, шляху до компілятора та появу елементів інтерфейсу. Друга містить налаштування,

що стосуються редактору коду, наприклад, шрифт коду та його розмір, автодоповнення дужок, перевірку парності дужок та інші. Обов'язковою умовою правильності шляху до компілятора має бути наявність файлу Asampl.exe в директорії компілятора. Після перевірки всіх налаштувань та їх застосування, нові налаштування записуються у форматі XML до файлу config.asamplefile.

При зміні налаштувань кольорової схеми викликається метод *SetAppearances* класу *MainForm* в якому обрана кольорова схема застосовується до всіх класів проекту, що успадковані від стандартних класів бібліотеки Winforms. Для більшості з цих класів визначені параметри прозорості, фонових кольорів, кольорів тексту. Для рендеру цих елементів використовуються класи, успадковані від стандартних класів. Для прикладу, на рисунку 3 зображено ієрархію успадкування від стандартного класу *ToolStripRenderer*.

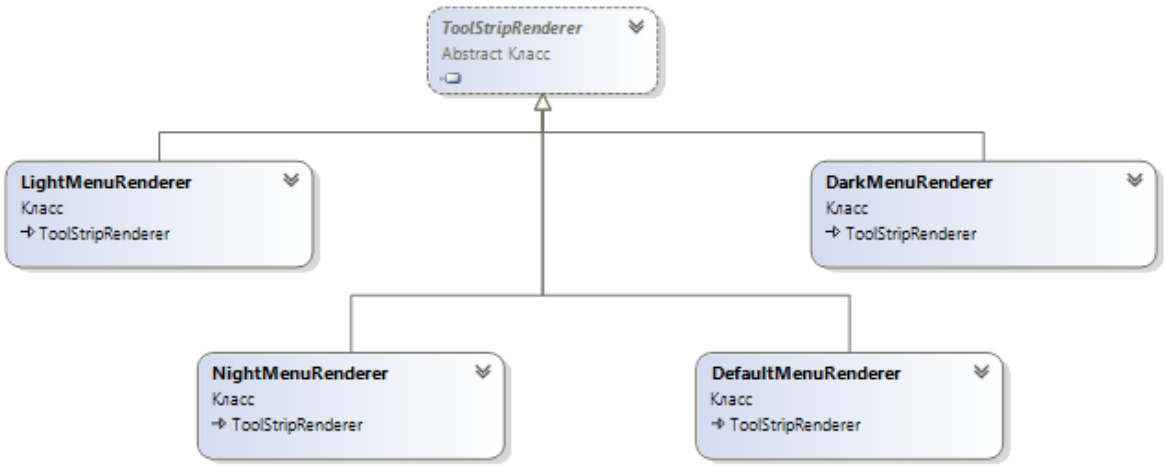


Рис. 3. Ієрархія успадкування класів від класу *ToolStripRenderer*

Аналогічною є ієрархія класів, що успадковуються від класу *ToolStripProfessionalRenderer*. Дані класи перевизначають методи рендеру елементів меню, розділювачів, елементів панелі інструментів, границь елементів та кнопок. Застосовуються класи рендеру для таких елементів головної форми: панелі меню, контекстного меню редактору, панелі дерева

файлів проекту, панелі елементів, логу компіляції, вкладок коду та їх контроллеру.

Наступним елементом головного меню є панель інструментів. Панель інструментів представлена класом *PanelWithBorder*, що успадковано від класу *Panel* Winforms. Для відповідності вибору кольорової схеми в цьому класі були визначені параметри прозорості, кольору та ширини границь, кольорів для градієнту та кут градієнту, а також було перевизначено метод батьківського класу *OnPaint* для використання вищезгаданих параметрів.

Панель інструментів надає швидкий доступ до деяких функцій меню файлів, таких як створення нового проекту, відкриття файлів та проекту, зберігання файлів; а також меню редагування та запуску: копіювання та вставка тексту, відміна та повтор останніх дій, компіляція та запуск коду.

Панель дерева файлів проектів показує список файлів, що додані до проекту. Дерево файлів представлене класом *TreeView* Winforms. При відкритті чи створенні проекту, відповідні методи оновлюють стан дерева, використовуючи файли проектів, що містять інформацію про файли, що включені до проекту.

Панель логу компіляції слугує для виведення інформації про статус компіляції. Панель складається с неактивного елементу *RichTextBox*, який є контейнером для інформації про помилку, отриманої від компілятора, чи повідомлення про успішну компіляцію від середовища розроблення, та елементу *PanelZ,* що є наслідником класу *Panel* Winforms. Для цього класу також були визначені параметри для рендеру: кольори та кут градієнту, прозорість.

Основним елементом головного меню є редактор, що представлений екземпляром класу *MyTabControl*, успадкованим від класу *TabControl* Winforms. Цей клас створено для відповідності кольоровій схемі середовища розроблення, і для нього визначено наступні параметри: кольори активних та не активних вкладок, колір тексту, прозорість, кути градієнту.

MyTabControl є контейнером для екземплярів класу *MyTabPage*, що відображають файли у редакторі. Клас *MyTabPage* відповідає за обробку коду та навігацію по ньому, наявність деяких елементів інтерфейсу та їх графічне зображення. В налаштуваннях користувач може обрати функцію автокомпіляції, що буде використовуватись під час роботи екземплярів класу *MyTabPage*. В даному випадку, автокомпіляція є заміною препроцесору в компіляторах інших мов програмування. При внесенні змін до коду, система автоматично запускає компілятор мови ASAMPL та виводить помилки за наявності. Автокомпіляція відбувається в три кроки. Першим кроком є виклик функції *Process_CompileAsampl*, що записує зміни до файлу з кодом та викликає наступну функцію *CompileASAMPL*, до якої передає назву файлу та шлях до виконуючого файлу компілятору. Дана функція запускає ще одну функцію *Compile*, передаючи до неї назву файлу та робочої директорії, та записує результат роботи компілятора до логу компіляції, зчитуючи його з стандартного потоку виведення. Функція *Compile* відповідає безпосередньо за створення процесу для роботи компілятора, перенаправляє потоки виведення та помилок так, щоб система могла їх прочитати та повертає статус запуску процесу – виникли помилки при його створенні чи ні.

Ще одним елементом головного меню є панель статусу, що представлена класом *StatusStrip* Winforms. Якщо в середовищі розроблення відкрито проект, панель статусу буде відображати назву поточного проекту, відритий файл та координати каретки в редакторі.

3.2. Особливості реалізації засобів автодоповнення та синтаксичного аналізу коду

Для реалізації підсвічення синтаксису було використано бібліотеку ICSharpCode.TextEditor. Підсвічення синтаксису в цій бібліотеці реалізовано класом *DefaultHighlightingStrategy*, що виконує підсвічення на основі XML-файлів із розширенням *.xshd*. В бібліотеку вбудовано багато

файлів із подібним розширенням, що описують синтаксис різних мов програмування, та автоматично співставляють розширення файлів с вихідним кодом та файл із синтаксисом.

Лістинг 3. Метод *AddLanguages* класу *MyTabPage*

```
public void AddLanguages(String lang){
    string dirc = Application.StartupPath;
    FileSyntaxModeProvider fsmp;
    if (Directory.Exists(dirc))
    {
        fsmp = new FileSyntaxModeProvider(dirc);
        HighlightingManager.Manager.
          AddSyntaxModeFileProvider(fsmp);
        textEditor.SetHighlighting(lang);
    }
    SetFont();
    SetViews();
    SetBackColorToTextEditor();}
```

Файл із синтаксисом мови ASAMPL описано у файлі Asampl.xshd, що має знаходитись поруч із виконуваним файлом проекту. Для опису синтаксису використовується XML, який оброблюється парсером бібліотеки і має містити відповідні елементи. Кореневим елементом такого файлу є *SyntaxDefinition* що містить назву мови та розширення файлів з кодом.

Опис правил синтаксису має знаходитись всередині елемента *RuleSets*. Цей елемент може включати один чи декілька наборів правил *RuleSet*. Такі елементи можуть мати атрибути. Наприклад, атрибут *ignorecase*, що може мати значення "true" чи "false", визначає чи буде ігноруватись регістр слів при виконанні цього правила. Іншим прикладом правила може бути визначення коментарів у коді:

```
<Span name="LineComment" rule ="CommentMarkerSet" bold="false"
       italic="false" color="Green" stopateol="true" >
   < Begin >//</Begin>
 </Span>
```

В даному прикладі визначений однолінійний коментар, що починається з символів "//", закінчується на символі закінчення строки, що вказано значенням "true" атрибуту *stopateol*, виділяється зеленим кольором без виділення курсивом та жирним шрифтом.

Ключові слова визначаються за допомогою елемента *KeyWords* та його дочірніх елементів *Key* з атрибутом *Word*. В даному прикладі створюється правило підсвічення з назвою *Operators*, та описує ключові слова, що виділяються зеленим кольором без виділення курсивом та жирним шрифтом:

```
<KeyWords name="Operators" bold="false" italic="false" color="Green">
            <Key word = "is" />
            <Key word = "from" />
            <Key word = "with" />
</KeyWords>
```

Також бібліотека підтримує обробку коду препроцесором, але через відсутність останнього в поточній версії компілятора мови ASAMPL, використання цієї функції не є актуальним.

Автодоповнення в системі представлене наступними функціями:

- доповнення коду на основі ключових слів та бібліотек;
- доповнення дужок та лапок;
- вставка шаблонів синтаксичних конструкцій.

При введені будь-якого символу в текстовому редакторі буде запущений процес автодоповнення, що буде виконуватись за умови, що дана функція ввімкнена в налаштуваннях редактора коду.

Автодоповнення коду ключовими словами працює наступним чином: серед файлів налаштувань завантажується файл keywords.asamplfile, де зберігаються ключові слова представлені в мові у форматі XML. Для зручності та пришвидшення пошуку відповідності слова зберігаються у список та сортуються. Список слів для доповнення спочатку очищується, а потім формується із ключових слів що можуть відповідати користувацькому вводу. Якщо даний список не є пустим, для введення користувачу пропонується перше слово із цього списку. Список відображається елементом *ListBox* для якого задається розмір відповідно обраному у системі розміру шрифту. *ListBox* розміщується в тому ж місці, де знаходиться каретка в редакторі коду, та отримує фокус.

Далі відбувається перевірка слова, введеного користувачем, на відповідність одному із шаблонів синтаксичної конструції в методі *ProcessToolTips*.

Лістинг 4. Метод *ProcessToolTips*

```
public void ProcessToolTips(String input){
      switch (input){
          case "else":
              ToolTipControl.ToolTipText = "else\nCode snippet for
                  else statement\n\nPress F2 to insert snippet";
              ToolTipControl.Width = 220;
              ToolTipControl.Height = 70;
              break;

          case "for":
              ToolTipControl.ToolTipText = "for loop\nCode snippet
                   for 'for' loop\n\nPress F2 to insert snippet";
              ToolTipControl.Width = 220;
              ToolTipControl.Height = 70;
              break;

          case "if":
              ToolTipControl.ToolTipText = "if\nCode snippet for if
                   statement\n\nPress F2 to insert snippet";
              ToolTipControl.Width = 220;
              ToolTipControl.Height = 70;
              break;

          default:
              if (isToolTipControlAdded)
                  ToolTipControl.Visible = false;
              break;}}
```

У випадку, якщо така відповідність встановлена, для елементу *ToolTipControl*, що відповідає за демонстрацію підказок, встановлюється текст та розміри вікна для нього. Для вставки шаблону синтаксичної конструції користувач має натиснути клавішу "F2". При цьому викликається метод *InsertingCodeSnippets*, що формує текст відповідного шаблону синтаксичної конструції в тимчасовій змінній, яка передається далі в метод *InsertSyntax*. Цей метод працює безпосередньо з текстовим редактором, замінюючи користувацький ввід на шаблону синтаксичної конструції та встановлюючи каретку редактору на кінець вставленого тексту. Останнім кроком роботи методу є видалення підказок та списків автодоповнення з вікна редактору.

Автодоповнення може бути вимкнене у налаштуваннях редактора коду.

3.3. Особливості реалізації відлагоджувача

Відлагоджування процесу починається із його запуску в дебаґері за допомогою функції *CreateProcess*, в параметрі *dwCreationFlags* якої вказується прапор *DEBUG_ONLY_THIS_PROCESS*, який вказує на відлагодження вказаного процесу але не процесів, що ним породжуються.

Головною частиною дебаґера є відлагоджувальний цикл, який базується на використанні функції *WaitForDebugEvent*. Одним із параметрів цієї функції є структура *DEBUG_EVENT*, що включає відлагоджувальну інформацію: ідентифікатор процесу, ідентифікатор потоку, код події та інформацію про неї. Коли функція *WaitForDebugEvent* повертає управління, потрібно обробити подію відлагоджувача та надати сигнал операційній системі для продовження обробки подій. Для цього необхідно викликати функцію *ContinueDebugEvent*, яка приймає параметрами ідентифікатори процесу та потоку, а також прапор, що визначає подальшу роботу дебаґера. Цей прапор може приймати два значення *DBG_CONTINUE* та *DBG_EXCEPTION_NOT_HANDLED*. Перше варто використовувати, коли виняткова ситуація, що виникла у відлагоджуваному процесі оброблена і ніщо не заважає подальшому виконанню. Друге значення свідчить про те, що виняткова ситуація не оброблена або не може бути оброблена відлагоджувачем. В такому випадку дебаґер може лише зробити запис про наявність такої виняткової ситуації.

Однією з подій, що обробляється реалізованим дебаґером є *OUTPUT_DEBUG_STRING_EVENT*, що використовується для виведення тексту. Інформація про цю подію зберігається в наступній структурі:

```
struct OUTPUT_DEBUG_STRING_INFO {
LPSTR lpDebugStringData;
WORD fUnicode;
WORD nDebugStringLength; };
```

Поле *nDebugStringLength* зберігає довжину строки. Поле *fUnicode* приймає значення "0", якщо строка формату ANSI, та відмінне від "0" значення, якщо формату Unicode. У другому випадку кількість зчитаних байт повинна бути вдвічі більшою за *nDebugStringLength*. Поле lpDebugStringData зберігає вказівник на рядок з повідомлення дебаґера, посилаючись на дані процесу, що відлагоджується.

Для зчитування пам'яті процесу, що відлагоджується, використовується функція ReadProcessMemory, що приймає на вхід ідентифікатор процесу, вказівник на пам'ять, що має бути зчитана, вказівник на пам'ять, куди має бути зчитане повідомлення та довжину зчитуваного повідомлення.

Лістинг 5. Обробка події *OUTPUT_DEBUG_STRING_EVENT*

```
case OUTPUT_DEBUG_STRING_EVENT:
      OUTPUT_DEBUG_STRING_INFO & DebugString =
                                    debug_event.u.DebugString;
      WCHAR *message=new WCHAR[DebugString.nDebugStringLength];
      ZeroMemory(message, DebugString.nDebugStringLength);
      ReadProcessMemory(processInfo.hProcess,
                        DebugString.lpDebugStringData,
                        message,
                        DebugString.nDebugStringLength, NULL);

      if (DebugString.fUnicode)
            stringEventMessage = message;
      else
            stringEventMessage = (LPSTR)message;
      delete []message;
      break;
```

Наступною подією, що оброблюється відлагоджувачем є *CREATE_PROCESS_DEBUG_EVENT*. Ця подія має бути першою обробленною дебаґером, оскільки виникає при створенні процесу. Структура *CREATE_PROCESS_DEBUG_INFO* зберігає інформацію про створений процес, таку як дескриптор виконуваного файлу, дескриптор процесу та головного потоку процесу. Для отримання назви файлу, що виконується використовується наступна функція:

```
GetFileNameFromHandle(debug_event.u.CreateProcessInfo.hFile);
```

Подія *LOAD_DLL_DEBUG_EVENT* викликається, коли операційна система завантажує динамічну бібліотеку, незалежно від того, відбувається це в коді явно чи ні. Структура *LOAD_DLL_DEBUG_INFO* зберігає інформацію про час завантаження та віртуальну адресу бібліотеки. Для отримання назви файлу бібліотеки теж використовується функція *GetFileNameFromHandle*.

При вивантаженні динамічних бібліотек викликається подія *UNLOAD_DLL_DEBUG_EVENT*, яка надає лише інформацію про віртуальну адресу бібліотеки, що не є інформативним. Тому для зберігання назв бібліотек використано структуру даних *Map*, внесення даних до якої потрібно додати до обробника події завантаження бібліотек. Таким чином отримаємо наступний код обробників подій:

Лістинг 6. Обробники подій завантаження/вивантаження бібліотек

```
std::map<LPVOID, CString> DllMap;
case LOAD_DLL_DEBUG_EVENT:
      stringEventMessage =
                  GetFileNameFromHandle(debug_event.u.LoadDll.hFile);

      DllMap.insert(std::make_pair(
                        debug_event.u.LoadDll.lpBaseOfDll,
                        stringEventMessage));

      stringEventMessage.AppendFormat(L"%x",
                        debug_event.u.LoadDll.lpBaseOfDll);
      break;

case UNLOAD_DLL_DEBUG_EVENT:
      stringEventMessage.Format(L"%s",
                        DllMap[debug_event.u.UnloadDll.lpBaseOfDll] );
      break;
```

Дві наступні події викликаються, коли створюється чи закінчується дочірний потік програми. *CREATE_THREAD_DEBUG_EVENT* надає інформацію про дескриптор потоку та стартову адресу потоку в пам'яті відлагоджуваного процесу. При обробці цієї події дана інформація надається користувачу. Варто зазначити, що при ініціалізації процесу дана подія не викликається для створеного головного потоку. Аналогічно, подія

EXIT_THREAD_DEBUG_EVENT виникає при завершенні дочірнього потоку та включає інформацію про код завершення потоку та ідентифікатор потоку, що використовується при обробці.

Останньою оброблюваною подією є *EXCEPTION_DEBUG_EVENT*, що генерується при виникненні виняткової ситуації у відлагоджуваному процесі. Структура *EXCEPTION_DEBUG_INFO* зберігає інформацію про виняткову ситуацію, що виникла, у структурі *EXCEPTION_RECORD*, та поле dwFirstChance. Деякі бібліотеки генерують first chance exception для відлагоджувачів, оскільки вони першими перехоплюють такі виняткові ситуації та можуть обробити їх швидше за процес, що відлагоджується. Виняткові ситуації, що генеруються відлагоджуваним процес, розцінюються як second chance exception, та можуть бути обробленими чи ні у відлагоджувачі.

В обробнику події виникнення виняткової ситуації відбувається вивід інформації про ситуацію, у вигляді коду, адреси та належності виняткової ситуації до типу first chance exception. Також встановлюється прапор *DBG_EXCEPTION_NOT_HANDLED*, що сигналізуватиме циклу дебаґера про завершення, у зв'язку з необробленим винятком.

Лістинг 7. Обробник події виникнення виняткової ситуації

```
case EXCEPTION_DEBUG_EVENT:
      EXCEPTION_DEBUG_INFO& exception = debug_event.u.Exception;
      switch(exception.ExceptionRecord.ExceptionCode){
            case STATUS_BREAKPOINT:
                  stringEventMessage= "Break point";
                  break;
            default:
                  if(exception.dwFirstChance == 1)
                        stringEventMessage.Format(L"First chance
                        exception at %x, exception-code: 0x%08x",
                        exception.ExceptionRecord.ExceptionAddress,
                        exception.ExceptionRecord.ExceptionCode);
                  else
                        stringEventMessage.Format(L"Second chance
                        exception at %x, exception-code: 0x%08x",
                        exception.ExceptionRecord.ExceptionAddress,
                        exception.ExceptionRecord.ExceptionCode);

                  dwContinueStatus = DBG_EXCEPTION_NOT_HANDLED;}
```

3.4. Висновки до розділу 3

В даному розділі було описано архітектуру програмного застосунку. Описано реалізації основних елементів інтерфейсу та функції, що пов’язані з ними.

Також було розглянуто та наведено реалізацію інструментів автодоповнення, синтаксичного аналізу коду, інструментів відлагодження коду.

4. АНАЛІЗ РОЗРОБЛЕНОГО ЗАСТОСУНКУ

4.1. Практичне застосування розробленого застосунку

Середовище розроблення програм на мові ASAMPL призначене для спрощення та прискорення процесу написання коду за допомогою таких функціональних можливостей:

- автодоповнення коду;
- підсвічення синтаксису;
- запуску компілятора безпосередньо з редактору;
- автокомпіляції коду;
- запуску відлагоджувача.

Реалізовані функції є основою більшості сучасних інтегрованих середовищ розроблення програмного забезпечення і роблять роботу з програмним кодом більш зручною, ніж прості текстові редактори.

На практиці розроблений застосунок можна використовувати для написання та відлагодження програмних застосунків мовою ASAMPL. На даний момент компілятор мови ASAMPL підтримує лише написання невеликих файлів коду, що можуть реалізовувати обробку відео та аудіо, тому не є можливим створення великих за обсягом проектів. Середовище розроблення підтримує створення робочих просторів у вигляді проектів, де розробник матиме змогу працювати з декількома файлами, пов'язаними чи не пов'язаними за змістом.

4.2. Реалізовані функціональні можливості

При розробленні IDE для мови ASAMPL, враховувалися архітектура інтерфейсу та функціональні можливості аналогічних систем, які в більшості використовують схожу модель взаємодії з користувачем.

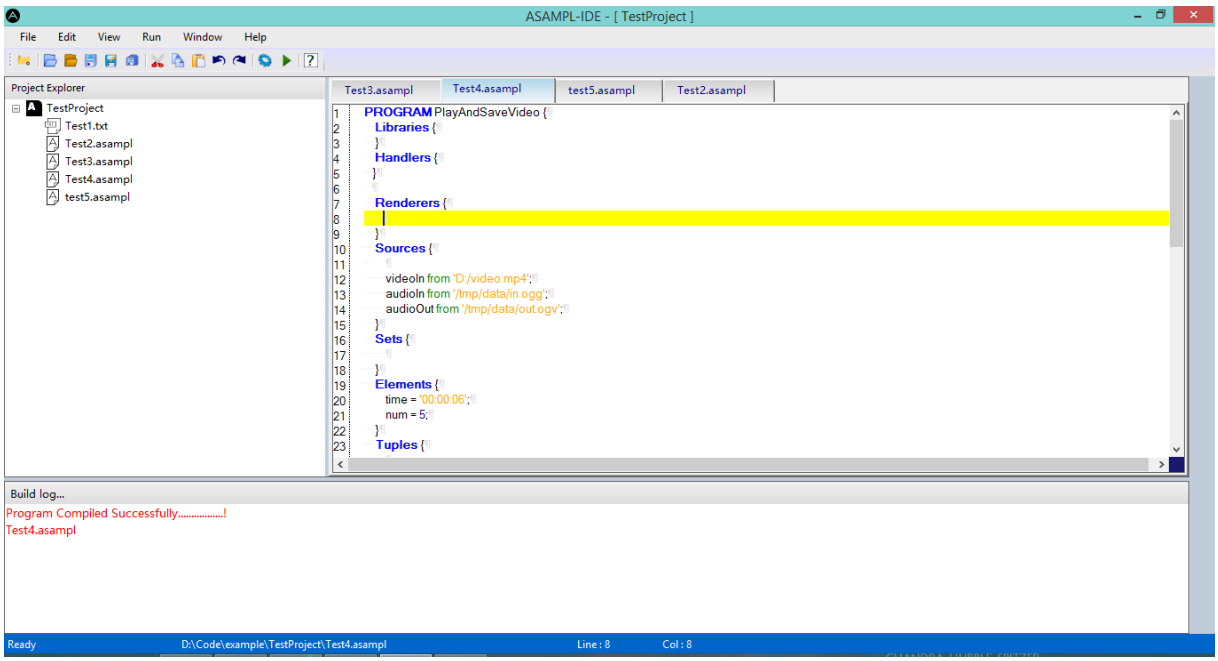


Рис. 4. Головне вікно програми

Головне вікно програми (рис. 4) складається з шести елементів:

- панелі меню;
- панелі інструментів;
- дерева файлів проекту;
- редактору коду;
- логу компіляції;
- панелі статусу.

Панель меню складається з шести меню. Меню *File* надає користувачу доступ до створення окремих файлів та проектів, їх збереження, закриття, видалення, а також дозволяє відкрити останній завантажений проект без пошуку його у файловій системі.

Пункт цього меню *New* дозволяє користувачу створити новий проект, текстовий файл, файл коду ASAMPL та файл із довільним розширенням. Якщо в редакторі не відкрито жодного проекту, надається лише можливість створення нового проекту, але не файлів. Створення нових файлів можливо лише у відкритому проекті, оскільки при цьому відбувається запис шляху

до створеного файлу до файлу проекту, і при його відсутності система генерувала б помилки.

Меню *Edit* представляє інтерфейс для роботи з файлами вихідного коду, що реалізовує наступні функції:

- вирізання, копіювання та вставка тексту;
- відкат чи повтор останньої дії;
- пошук та пошук з заміною в тексті;
- перехід до вказаної строки коду;
- видалення виділеного тексту;
- виділення всього тексту;
- зміна регістру виділеного тексту на верхній чи нижній;
- вставка коментарів – однолінійного та мультилінійного.
- навігація між документами.

Функція відкату та повтору останньої дії має необмежений буфер, тобто зберігає всі зміни, починаючи з моменту відкриття файлу в редакторі. Для пошуку та пошуку з заміною відкриваються нові вікна. При використанні цих функцій варто враховувати, що вони є регістрозалежними, тому результати пошуку, наприклад, слів "From" та "from", будуть відрізнятися.

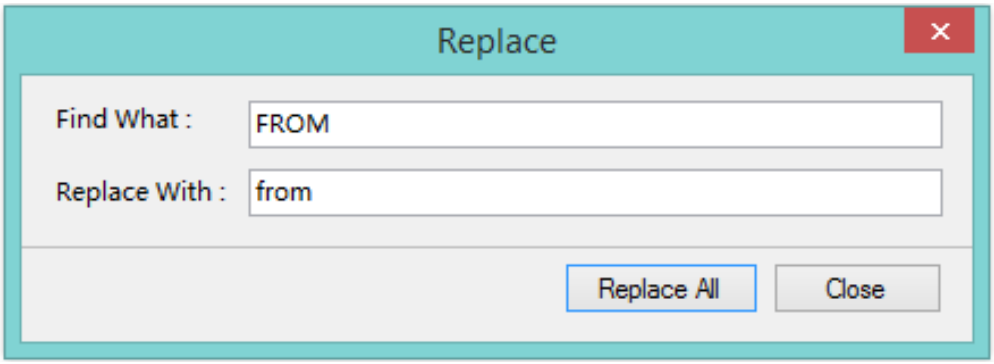


Рис. 5. Приклад роботи форми для заміни тексту

Меню *View* дозволяє користувачеві налаштовувати вигляд середовища розроблення. Пункт меню *Tabs Alignment* відповідає за

розташування списку відкритих вкладок: над робочою зоною текстового редактора чи під нею. Пункти *Status Strip*, *Tool Strip*, *Project Explorer* та *Build Log* дозволяють користувачу показувати чи ховати відповідні елементи інтерфейсу головного вікна. Також в меню *View* є можливість переведення головного вікна в повноекранний режим, швидкого доступу до зміни кольорової схеми середовища та деяких налаштуваннях текстового редактору, таких як показ табуляцій та символів закінчення рядку, нумерації рядків та виділення кольором поточного рядку. При зміні налаштувань текстового редактору буде виведене повідомлення про необхідний перезапуск середовища розроблення для того, аби вказані налаштування були застосовані.

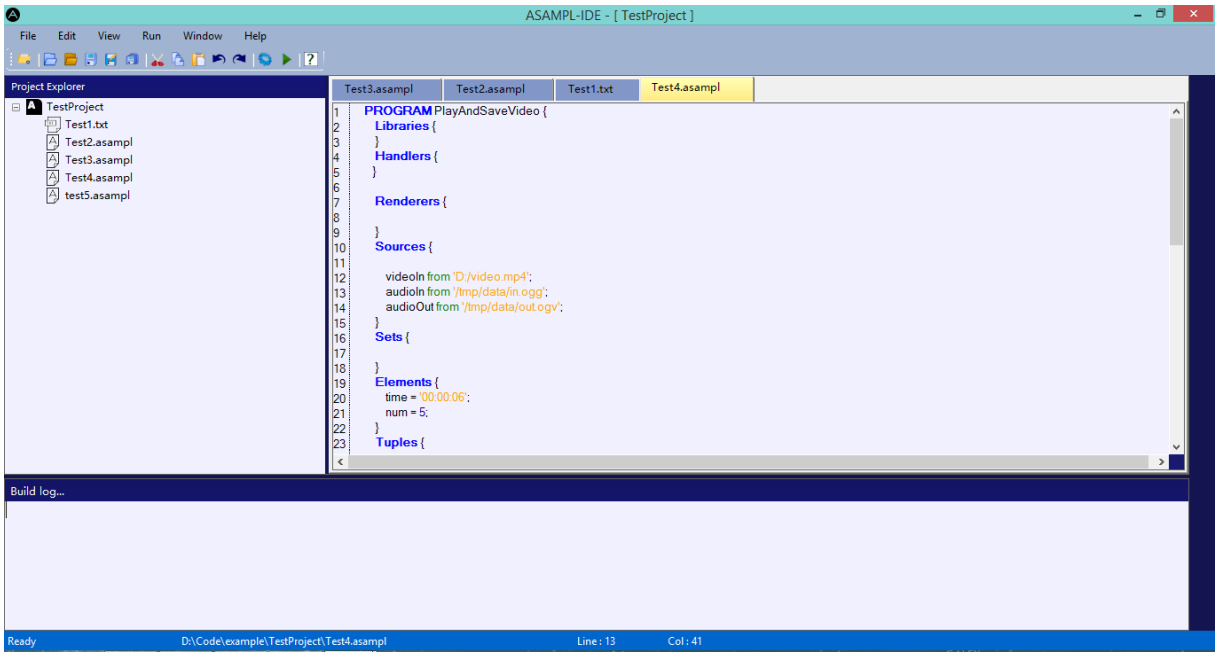


Рис. 6. Приклад роботи програми з темною кольоровою схемою

Меню *Run* надає доступ до функцій компіляції, запуску та відлагодження програм. Дані функції працюють лише при завантаженому проекті з файлом, що відкритий у поточній активній вкладці редактору коду. Оскільки поточна версія компілятора ASAMPL не генерує виконуваних файлів, компілює поданий на вхід файл та одразу його виконує, при використанні функції компіляції в середовищі розроблення, результати

роботи компілятора будуть виведені у вікно логу компіляції, після чого код програми буде виконано. У випадку успішної компіляції буде виведено відповідне повідомлення, інакше – список помилок. При наявності помилок редактор автоматично переставляє каретку та виділяє лінію з помилкою, вказаною в компіляторі. Особливістю реалізації поточної версії компілятора є те, що він обробляє лише одну помилку за цикл своєї роботи. Тобто при наявності декількох помилок в коді, компілятор має бути запущений декілька раз для виявлення всіх помилок. Також меню *Run* надає можливість обрати опцію автокомпіляції. Функція автокомпіляції викликає компілятор кожного разу, коли вносяться зміни до файлу в редакторі коду. При цьому вікна, що можуть бути створені компілятором для виконання цього коду ховаються і лише до логу компіляції записується результат роботи. Дану функцію можна назвати заміною прекомпіляції коду.

Ще однією функцією доступною в меню *Run* є доступ до налаштувань середовища розроблення. При цьому з'являється відповідне вікно налаштувань, що містить дві вкладки: налаштування середовища та налаштування редактору коду. Опції, що доступні для вибору, є аналогічними до тих, що доступні в меню View. Окрім цього, доступний вибір шляху до виконуваного файлу компілятора. Для цього необхідно вибрати директорію, у якій знаходиться цей файл, що обов'язково має мати назву *Asampl.exe*. В іншому випадку, користувачу буде виведене повідомлення про помилку пошуку компілятора у вказаній директорії. При першому запуску середовища розроблення користувачу також буде виведене вікно з вимогою встановити шлях до виконуваного файлу компілятора. У випадку, якщо цей шлях буде некоректним та не міститиме необхідного файлу – середовище розроблення закінчить роботу.

Меню *Window* дозволяє швидко перезапустити середовище розроблення, закрити всі вкладки з кодом та перейти до кожної з відкритих вкладок.

Меню *Help* надає довідкові дані про роботу системи.

Наступним елементом інтерфейсу є панель інструментів, що надає швидкий доступ до часто використовуваних функцій, таких як створення нового проекту, відкриття існуючого проекту та файлів, зберігання вибраного файлу, зберігання зі зміною назви, зберігання всіх файлів, копіювання, вирізання та вставка тексту, відміна та повтор останньої дії, компіляція та запуск поточного файлу, доступ до довідкової інформації.

Дерево файлів проекту (рис. 7) зберігає та виводить користувачу всі файли, що вказані у файлі проекту. Користувач має можливість відкривати ці файлі в редакторі коду. При натисканні правою кнопкою миші на дереві проекту, виводиться контекстне меню, що дозволяє створити нові файли та додати їх до проекту, відкрити вказаний файл у редакторі чи з використанням редактору встановленого системою, видалити файл та закрити проект.

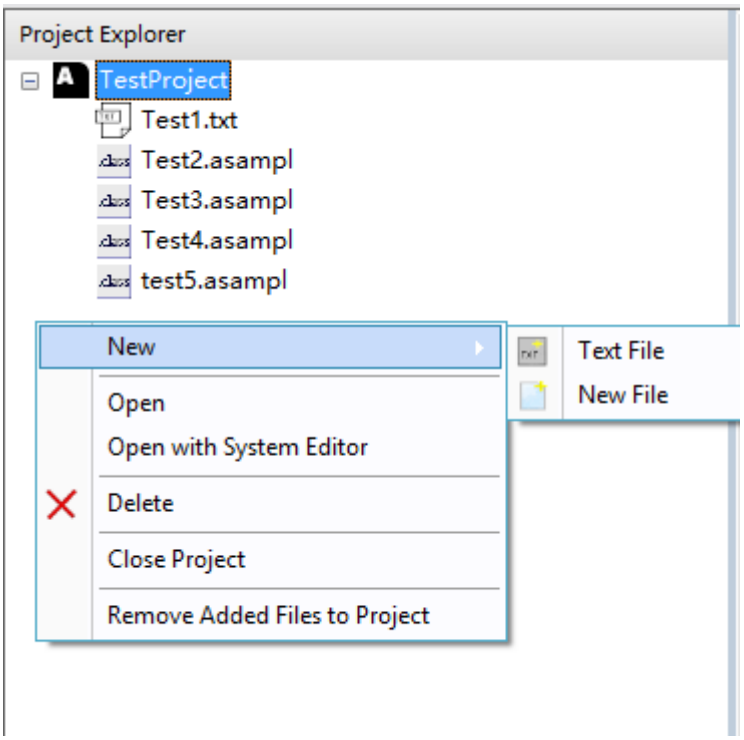


Рис. 7. Дерево файлів проекту

Основним елементом середовища розроблення є редактор коду. Панель вкладок у верхній частині редактору (чи нижній, якщо це вибрано в налаштуваннях системи) дозволяє користувачу переходити між відкритими

у редакторі файлами. При натисканні правою кнопкою миші на назві вкладки відкривається контекстне меню, що дозволяє зберегти вибраний файл, закрити його, закрити всі вкладки (з можливістю залишити обрану вкладку відкритою). Також користувач може скопіювати повний шлях файлу у файловій системі до буферу або відкрити директорію, у якій знаходиться вибраний файл.

При запуску системи в редакторі створюється стартова сторінка, що містить декілька рядків тексту, що виглядають як гіперпосилання. Ці рядки дозволяють користувачу викликати функції створення нового проекту, відкриття існуючого проекту чи файлів. У налаштуваннях середовища розроблення можна приховати цю сторінку при кожному новому запуску редактора.

Редактор коду також має контекстне меню, що надає доступ до деяких функцій роботи з текстом, таких як копіювання, вставка, вирізання, видалення, виділення всього тексту, зміна регістру тексту та вставка коментарів.

Як було сказано раніше в цьому розділі, існують деякі налаштування текстового редактору, доступні для зміни. Окрім звичних налаштувань шрифту та його розміру, користувач може прибрати панель з нумерацією рядків коду, виділення кольором рядку коду, в якому знаходиться каретка редактору, наявність зображуваних символів табуляції та закінчення рядку. Серед інших налаштувань є декілька, що стосуються автодоповнення коду. Ці налаштування відповідають за автоматичне доповнення дужок у коді, а також за співставлення відкриваючої та закриваючої дужки у коді, шляхом виділення їх кольором.

Автодоповнення коду працює для ключових слів, назв бібліотек та деяких синтаксичних конструкцій. При написанні слова в редакторі з'являється меню вибору варіантів закінчення, що були знайдені в списку ключових слів, що зберігається в файлах налаштувань проекту. Для навігації між ними використовуються клавіши зі стрілками, а для вибору слова –

клавіша Enter. Система підсвічення коду (рис. 8) виділяє синім кольором та жирним шрифтом ключові слова програми (*Libraries*, *Program* та інші), зеленим кольором ключові слова операторів (*is*, *from*, *download* та інші), помаранчевим – текст, що знаходиться в лапках, та зеленим -коментарі. Не дивлячись на те, що система підтримує як однолінійні, так і багатолінійні коментарі, компілятор поточної версії підтримує лише однолінійні. Користувач може налаштувати систему підсвічення синтаксису власноруч, відредагувавши файл Asampl.xshd, що знаходиться поруч із виконуваним файлом проекту.

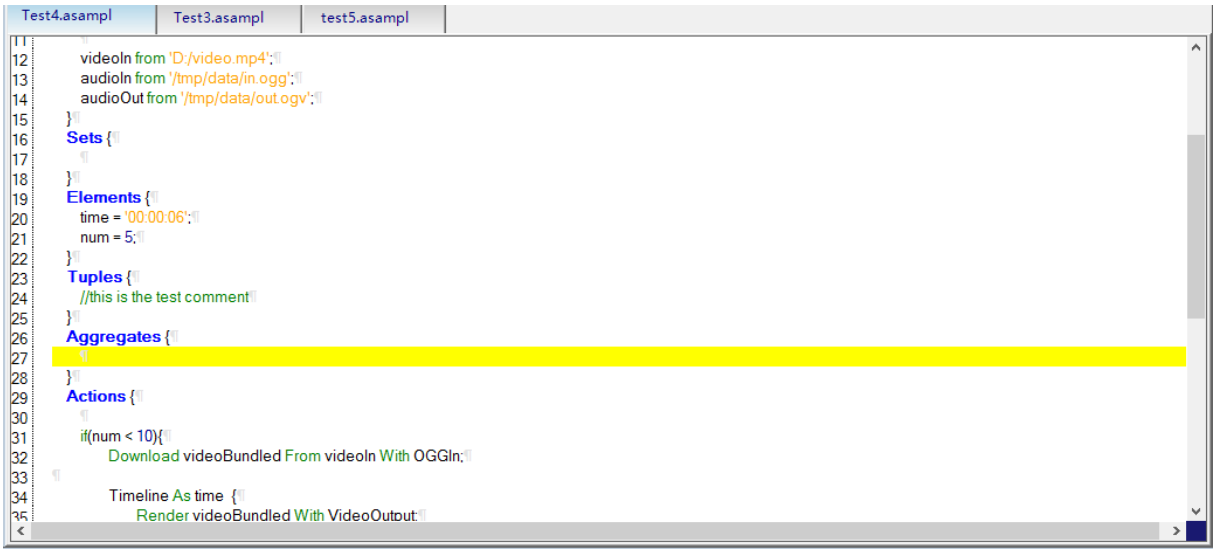


Рис. 8. Приклад роботи системи підсвічення коду

Лог компіляції виводить інформацію про перебіг компіляції коду. Користувач може отримати повідомлення про успішний результат роботи компілятора, або повідомлення з помилкою, яку видає компілятор при обробці даного сегменту коду. Повідомлення компілятора містить номер рядку коду в якому очікувалась синтаксична структура відміна від тієї, що була подана на вхід. Користувач не може вносити зміни до тексту логу компіляції, але може очистити його та скопіювати текст помилки для подальшого його використання.

Панель статусу внизу головного вікна програми відображає інформацію про поточний відкритий проект та координати каретки в редакторі коду.

В рамках системи також реалізовано відлагоджувач. В поточній його версії реалізована обробка з виведенням інформації про події, що відбуваються під час роботи відлагоджуваного процесу, такі як завантаження динамічних бібліотек, створення дочірніх процесів та потоків, виникнення виняткових ситуацій.

4.3. Напрямки подальшого вдосконалення

Розроблена система знаходиться лише на початковому етапі свого розвитку. Тому подальших шляхів вдосконалення існує чимало.

Перш за все, розглянемо можливості текстового редактору. Сучасні IDE пропонують чимало засобів для рефакторингу коду та зручної навігації в коді. На даний момент в системі реалізована функція пошуку заданого тексту в коді. На її основі можливо реалізувати пошук всіх посилань на змінну чи функцію в коді. Функцію пошуку та заміни тексту в коді можна розширити за рахунок можливості ігнорування регістру, використання регулярних виразів, пошуку цілого слова чи його частини, застосування заміни до виділеної області, поточного документу чи всього проекту.

Також для спрощення навігації в коді варто було б реалізувати перехід до реалізації, ініціалізації чи визначення об'єктів, функцій чи змінних. Іншою зручною особливістю сучасних редакторів коду є згортання коду функцій, класів чи синтаксичних конструкцій.

Впровадження прекомпіляції в редактор коду системи значно б спростило роботу з кодом, за рахунок його аналізу ще на етапі написання. Поточна версія системи базується на бібліотеці ICSharpCode.TextEditor, яка підтримує використання прекомпіляторів для синтаксичного аналізу, але даний інструмент поки не реалізований в компіляторі ASAMPL.

Реалізація підсвічення синтаксису та автодоповнення може бути розширена новими правилами для аналізатору, списками додаткових ключових та шаблону синтаксичної конструції. Власне, робота з файлами, що описують синтаксис мови для аналізатору та ключові слова не потребує від користувача зміни коду, тому він може налаштовувати такі файли під власні потреби.

На даному етапі розвитку компілятор не генерує виконуваних файлів, виконуючи код одразу після компіляції. Тому необхідності в окремих функціях для запуску коду наразі немає. Але з розвитком компілятору, коли буде реалізоване розділення компіляції коду та його виконання, ці функції знову потрібні. В майбутніх версіях системи необхідно буде лише внести невеликі зміни до коду, аби реалізувати запуск виконуваного файлу окремо від процесу компіляції.

Одним із можливих напрямів розвитку є підтримка декількох мов програмування. Наразі більшість сучасних IDE підтримують декілька мов, надаючи необхідний інтерфейс одразу в системі чи за допомогою сторонніх плагінів. Підтримка користувацьких плагінів також може суттєво розширити функціональні можливості середовища розроблення. Однак в даному випадку варто було б перейняти принцип використовуваний у Eclipse IDE: "Плагіни для Eclipse розробляються в самій Eclipse". Тобто надати користувачам можливість розробляти плагіни в середині самого середовища розроблення. Для цього також необхідно буде створити відповідну документацію.

Одним із шляхів розвитку мови в цілому, що також вплине на розвиток середовища розроблення, є реалізація компонувальника – системи, що буде поєднувати декілька об'єктних файлів у один виконуваний. Всередині компілятора також необхідно буде реалізувати підтримку використання декількох файлів у проекті одночасно. На основі даних змін використання проектів у середовищі розроблення буде більш логічним, ніж

в поточній версії, де проект є скоріше робочою зоною, де користувач може створити декілька непов'язаних між собою файлів.

Оскільки ASAMPL є мовою для роботи з мультимодальними даними, було б доцільним відтворення цих даних всередині середовища розроблення. Це б зробило роботу з відео, аудіо чи іншими даними, більш зручною. Додаючи файли з мультимодальними даними до проекту, можна суттєво спростити доступ до них, не вказуючи повний шлях у файловій системі, а лише зробивши посилання на ресурси цього проекту.

Окрім вищезгаданих інструментів, для будь-якого середовища розроблення однозначним бонусом є підтримка систем контролю версій (Git, Mercurial) та командного розроблення (Team Foundation Server). Наразі впровадження таких рішень не виглядає актуальним, оскільки проекти мовою ASAMPL ще не мають великих обсягів коду, але з розвитком мови, так чи інакше, виникне питання про підтримку таких систем.

Однією з найскладніших систем будь-якої IDE є дебаґер. Відлагоджувач, реалізований в системі, є дуже обмеженим і відображає лише невелику кількість інформації про події, що виникають в процесі роботи програми, без можливості вплинути на цей процес. Розробленя дебаґеру є дуже складною задачею, оскільки потребує розуміння роботи процесів, потоків, подій на низькому рівні абстракції. Довідкової літератури з описом розроблення подібних систем досить мало, як і прикладів реалізації.

При розробленні наступних версій дебаґеру мають бути реалізовані наступні функціональні можливості:

- покрокове виконання коду;
- виконання коду до точки зупинки;
- доступ до змінних, що знаходяться в пам'яті відлагоджуваного процесу.

Описані напрямки розвитку можуть суттєво вдосконалити систему, наблизивши її до аналогічних середовищ розроблення для інших мов, а

також популяризувати мову ASAMPL за рахунок надання користувачам потужних засобів для роботи з нею.

4.4. Висновки до розділу 4

В даному розділі було описано практичне застосування розробки, реалізовані функціональні можливості, процес роботи з ними з точки зору користувача, наведено приклади роботи системи.

Було детально розглянуто напрямки подальшого розвитку розробленого застосунку, з метою розширення його функціональних можливостей, створення комфортних умов для роботи із системою.

ВИСНОВКИ

Метою даного дипломного проекту було створення середовища розроблення програм на мові ASAMPL.

В результаті аналізу існуючих середовищ розроблення для інших мов програмування, було сформовано основі вимоги до функціональних можливостей розроблюваного програмного застосунку. Проведені дослідження підтвердили актуальність потреби в створенні програмного рішення, що дозволяє спростити процес та скоротити час розроблення програм мовою ASAMPL.

Після розгляду поставленої задачі були виявлені основні вимоги до програмного продукту та засобів розроблення. На їх основі виконано вибір та порівняння наявних технологій, спроектована та реалізована архітектура середовища розроблення програм на мові ASAMPL. В основі розроблення лежать наступні технології: мова програмування C# та бібліотека Windows Forms для реалізації власне середовища розроблення, та мова програмування C++ для реалізації відлагоджувача.

Заплановано подальше розширення функціональних можливостей розробленого програмного застосунку з метою покращення зручності користування ним, прискорення та спрощення процесу написання коду.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Sulema, Y. ASAMPL: Programming Language for Mulsemedia Data Processing Based on Algebraic System of Aggregates [Text] / Yevgeniya Sulema ; ed. Auer M.E., Tsiatsos T. // Interactive Mobile Communication Technologies and Learning – Springer International Publishing, 2018. – pp. 431–442 – ISBN 978-3-319-75175-7.
2. Страуструп Б. Дизайн та еволюція C++ [Текст] / Бьйорн Страуструп – СПб.: Питер, 2007. – 445 с. – ISBN 5-469-01217-4.
3. Using ICSharpCode.TextEditor [Електронний ресурс] – Режим доступу: https://www.codeproject.com/Articles/30936/Using-ICSharpCode-TextEditor – Дата доступу: 14.04.2019.
4. 10 лучших IDE [Електронний ресурс] – Режим доступу: https://www.internet-technologies.ru/articles/10-luchshih-ide.html – Дата доступу: 03.05.2019
5. 10 Best Ide in the World [Електронний ресурс] – Дата доступу: 01.06.2019 – Режим доступу: https://medium.com/issuehunt/10-best-ide-in-the-world-c4875d1300c5 – Дата доступу: 01.06.2019
6. Syntax highlighting [Електронний ресурс] – Режим доступу: https://github.com/icsharpcode/SharpDevelop/wiki/Syntax-highlighting – Дата доступу: 15.05.2019
7. Writing a basic windows debugger [Електронний ресурс – Режим доступу: https://www.codeproject.com/Articles/43682/Writing-a-basic-Windows-debugger – Дата доступу: 20.05.2019
8. Texteditor FAQs [Електронний ресурс]– Режим доступу: https://github.com/icsharpcode/SharpDevelop/wiki/Texteditor-FAQs – Дата доступу: 19.05.2019

9. Best IDE Software FAQs [Електронний ресурс] – Режим доступу: https://www.g2.com/categories/integrated-development-environment-ide - – Дата доступу: 02.06.2019
10. Best IDE – 2018 For Programming (Top 12) [Електронний ресурс] – Режим доступу: https://techworm.net/programming/best-ide-programming-2018/ – Дата доступу: 01.05.2019
11. Популярные среды разработки и их недостатки [Електронний ресурс]– Режим доступу: https://geekbrains.ru/posts/ide_negative – Дата доступу: 03.05.2019
12. Описание среды разработки Visual Studio [Електронний ресурс] – Режим доступу: https://studbooks.net/2258619/informatika/opisanie_sredy_razrabotki_microsoft_visual_studio – Дата доступу: 29.05.2019
13. Обзор платформы Eclipse [Електронний ресурс] – Режим доступу: https://hightech.in.ua/content/art-eclipse-platform – Дата доступу: 29.05.2019
14. Winforms vs WPF [Електронний ресурс] – Режим доступу: https://www.educba.com/winforms-vs-wpf/ – Дата доступу: 14.05.2019
15. C++ Best GUI [Електронний ресурс] – Режим доступу: https://tproger.ru/digest/cpp-best-gui/ – Дата доступу: 07.05.2019
16. GUI на Java [Електронний ресурс] – Режим доступу: https://xakep.ru/2014/09/10/java-gui/ – Дата доступу: 08.05.2019
17. Windows Forms [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/dotnet/framework/winforms/– Дата доступу: 08.05.2019

18. Обратная разработка. Отладчик [Електронний ресурс] – Режим доступу: https://tgraph.io/Obratnaya-razrabotka-Otladchik-08-03 – Дата доступу: 17.05.2019
19. Debugging a running process [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/windows/desktop/debug/ debugging-a-running-process – Дата доступу: 17.05.2019
20. How debbugers really work process [Електронний ресурс] – Режим доступу: https://opensource.com/article/18/1/how-debuggers-really-work – Дата доступу: 05.05.2019
21. OpenJDK Wiki [Електронний ресурс] – Режим доступу: https://wiki.openjdk.java.net/display/OpenJFX/Main – Дата доступу: 01.04.2019
22. 9 Must Decisions in Desktop Application Development for Windows [Електронний ресурс] – Режим доступу: https://michaelscodingspot.com/9-must-decisions-in-desktop-application-development-for-windows/ – Дата доступу: 01.04.2019
23. Сравнение производительности UI в WPF, Qt, WinForms и FLTK [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/263897/ – Дата доступу: 01.04.2019
24. How to Create an IDE [Електронний ресурс] – Режим доступу: https://tekeye.uk/programming/how-to-create-an-ide– Дата доступу: 19.03.2019
25. Open files with the OpenFileDialog [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/enus/dotnet/framework/winforms/controls/how-to-open-files-using-the-openfiledialog-component– Дата доступу: 25.04.2019

26. Reading XML with the XmlReader class [Електронний ресурс] – Режим доступу: https://csharp.net-tutorials.com/xml/reading-xml-with-the-xmlreader-class/ – Дата доступу: 26.04.2019

27. Introduction to Winforms material design [Електронний ресурс] – Режим доступу: https://www.grapecity.com/blogs/winforms-material-design-introduction – Дата доступу: 26.04.2019

28. TreeView C# [Електронний ресурс] – Режим доступу: https://metanit.com/sharp/windowsforms/4.13.php– Дата доступу: 28.04.2019

29. WaitForDebugEvent function [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/windows/desktop/api/debugapi/nf-debugapi-waitfordebugevent– Дата доступу: 15.05.2019

30. C++ vs C# [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/c-vs-c-sharp/– Дата доступу: 15.05.2019

31. How to create an editor with syntax highlighting [Електронний ресурс] – Режим доступу: https://tomassetti.me/how-to-create-an-editor-with-syntax-highlighting-dsl/– Дата доступу: 15.05.2019

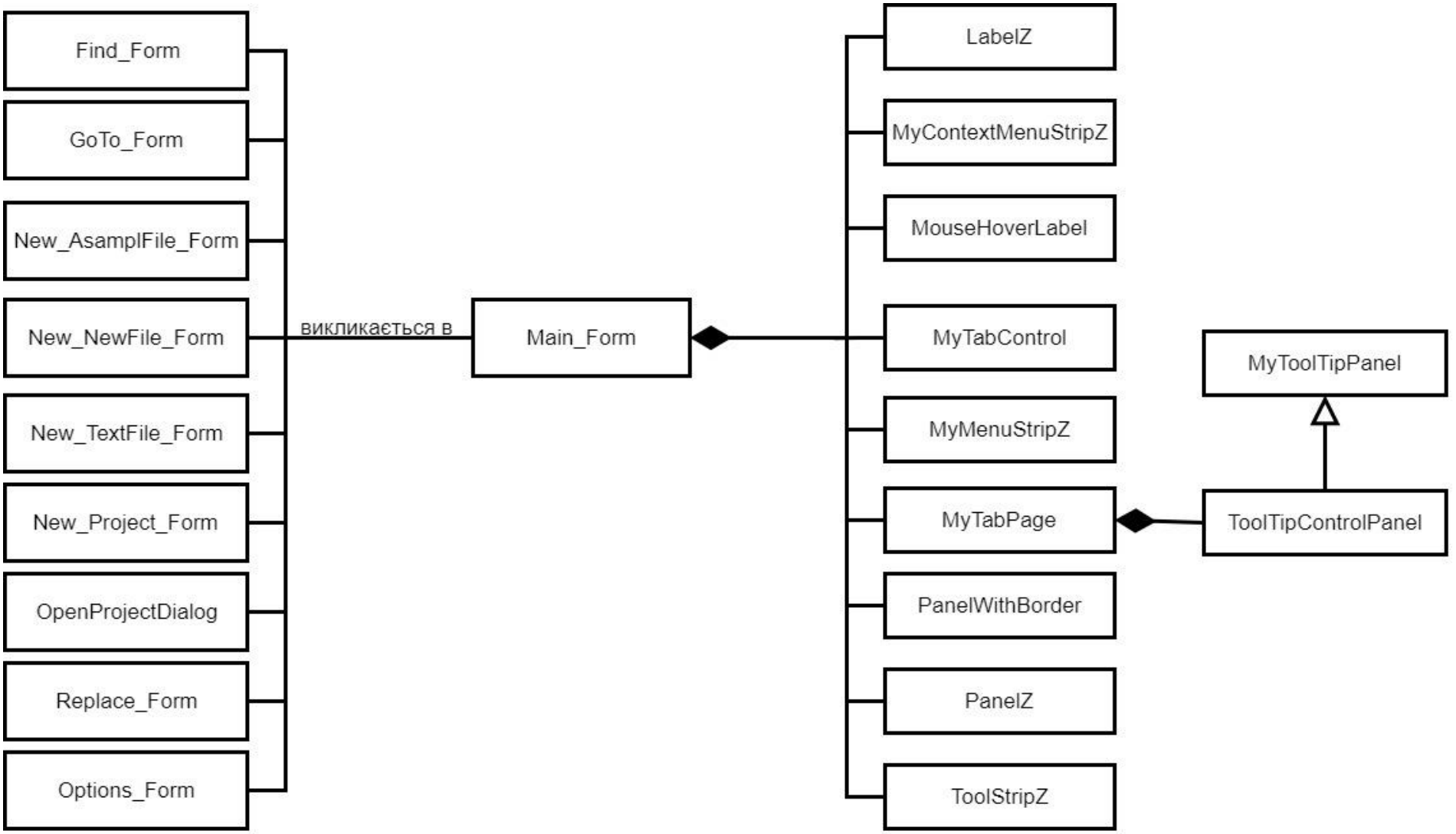
32. Syntax Highlighting Guide [Електронний ресурс] – Режим доступу: https://code.visualstudio.com/api/language-extensions/syntax-highlight-guide– Дата доступу: 14.05.2019

33. How does code completion works [Електронний ресурс] – Режим доступу: https://stackoverflow.com/questions/1220099/how-does-code-completion-work – Дата доступу: 14.05.2019

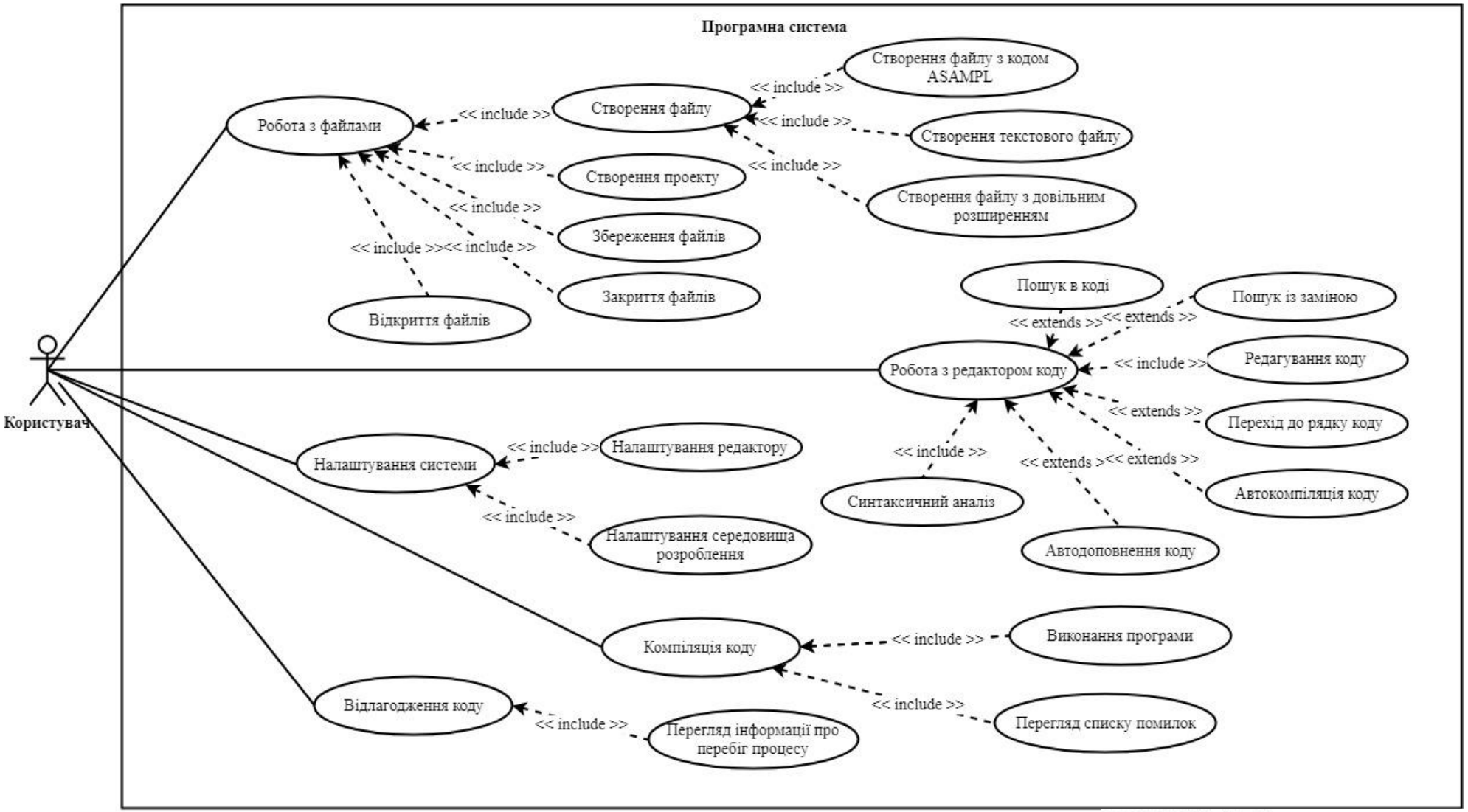
34. ICSharpCode.TextEditor TextEditorControl examples [Електронний ресурс] – Режим доступу https://csharp.hotexamples.com/ru/examples/ICSharpCode.TextEditor/TextEditorControl/-/php-texteditorcontrol-class-examples.html– Дата доступу: 14.05.2019
35. NuGet Gallery [Електронний ресурс] – Режим доступу: https://www.nuget.org/ – Дата доступу: 11.05.2019.
36. ISO/IEC 23270:2006 – Information technology – Programming languages– C# [Електронний ресурс] – Режим доступу: https://www.iso.org/standard/42926.html – Дата доступу: 11.05.2019
37. C++ [Електронний ресурс] – Режим доступу: https://en.cppreference.com/w/– Дата доступу: 09.05.2019
38. XSHD Tags [Електронний ресурс] – Режим доступу: https://github.com/icsharpcode/SharpDevelop/wiki/XSHD-Tags – Дата доступу: 09.05.2019
39. C# Process Examples [Електронний ресурс] – Режим доступу: https://www.dotnetperls.com/process– Дата доступу: 09.05.2019
40. Hiding and Showing forms in C# [Електронний ресурс] – Режим доступу:https://www.techotopia.com/index.php/Hiding_and_ Showing_Forms_ in_C_Sharp– Дата доступу: 09.05.2019

ДОДАТКИ

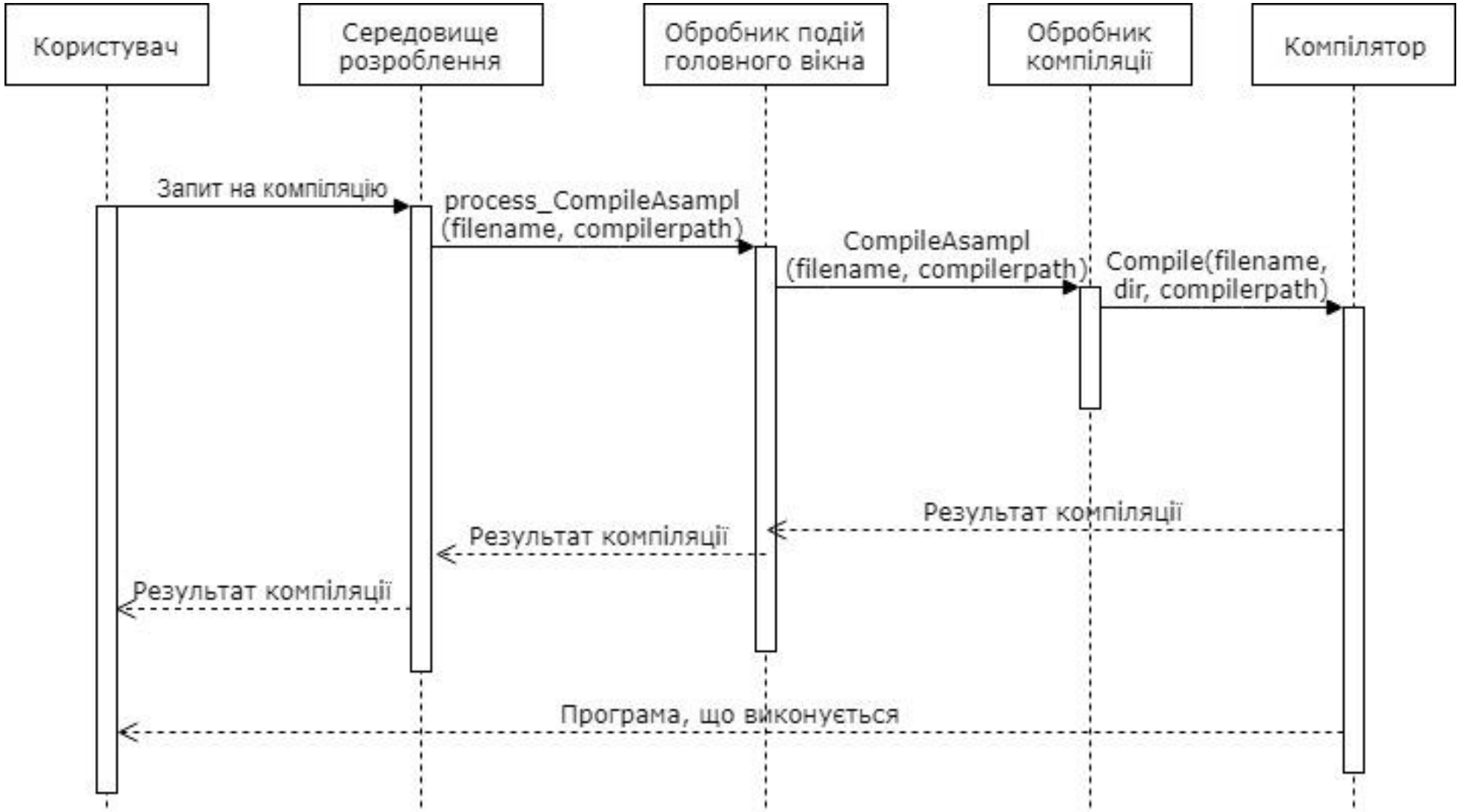
Додаток 1
Копії графічних матеріалів



ДП.045440-06-99
Середовище розроблення програм
на мові ASAMPL. Архітектура середовища
розроблення. Діаграма класів



ДП.045440-07-99
Середовище розроблення програм на мові ASAMPL. Функціональні можливості застосунку. Діаграма використання



ДП.045440-08-99
Середовище розроблення програм на мові ASAMPL. Процес компіляції. Діаграма послідовності

Приклад роботи системи підсвітки синтаксису.

```
PROGRAM PlayAndSaveVideo {
    Libraries {
    }
    Sources
        {
        videoIn from 'D:/video.mp4';
        audioIn from '/tmp/data/in.ogg';
        audioOut from '/tmp/data/out.ogv';
    }

    Elements {
        time = '00:00:06';
        num = 5;
}
    Actions {

        if(num < 10){
                Download videoBundled From videoIn With OGGIn;

                Timeline As time  {
                        Render videoBundled With VideoOutput;
                }
        }
    }
```

```
1   PROGRAM PlayAndSaveVideo {
2     Libraries {
3     }
4     Sources
5       {
6       videoIn from 'D:/video.mp4';
7       audioIn from '/tmp/data/in.ogg';
8       audioOut from '/tmp/data/out.ogv';
9     }
10
11    Elements {
12      time = '00:00:06';
13      num = 5;
14  }
15    Actions {
16
17      if(num < 10){
18          Download videoBundled From videoIn With OGGIn;
19
20          Timeline As time  {
21              Render videoBundled With VideoOutput;
22          }
23      }
24    }
```

Група КП-52, Крисюк А.М.

Додаток 2
Лістинг програми

Лістинг 1. Головний цикл відлагоджувача.

```
void CCrudeDebuggerDlg::DebuggerThreadProc()
{
	STARTUPINFO startupInfo;
	PROCESS_INFORMATION processInfo;

	ZeroMemory( &startupInfo, sizeof(startupInfo) );
	startupInfo.cb = sizeof(startupInfo);
	ZeroMemory( &processInfo, sizeof(processInfo) );

	CreateProcess(ProcessToDebug, NULL, NULL, NULL, false,
	DEBUG_PROCESS, NULL,NULL, &startupInfo, &processInfo);

	CString stringEventMessage;
	std::map<LPVOID, CString> DllMap;


	DEBUG_EVENT debug_event ={0};

	bool bContinueDebugging = true;

	DWORD dwContinueStatus = DBG_CONTINUE;

	while(bContinueDebugging){
	if (!WaitForDebugEvent(&debug_event, INFINITE))
		return;

	switch(debug_event.dwDebugEventCode){
	case CREATE_PROCESS_DEBUG_EVENT:
		stringEventMessage =
		GetFileNameFromHandle(debug_event.u.CreateProcessInfo.hFile);

		break;

	case CREATE_THREAD_DEBUG_EVENT:
		stringEventMessage.Format(L"Thread 0x%x (Id: %d) created at:
		0x%x",  debug_event.u.CreateThread.hThread,
		debug_event.dwThreadId,
		debug_event.u.CreateThread.lpStartAddress);
		break;
	case EXIT_THREAD_DEBUG_EVENT:

		stringEventMessage.Format(L"The thread %d exited with code:
		%d", debug_event.dwThreadId,
		debug_event.u.ExitThread.dwExitCode);
		break;

	case EXIT_PROCESS_DEBUG_EVENT:
		stringEventMessage.Format(L"0x%x",
		debug_event.u.ExitProcess.dwExitCode);
		bContinueDebugging=false;
		break;

	case LOAD_DLL_DEBUG_EVENT:
		stringEventMessage =
		GetFileNameFromHandle(debug_event.u.LoadDll.hFile);

		DllMap.insert(
			std::make_pair( debug_event.u.LoadDll.lpBaseOfDll,
			stringEventMessage));
		stringEventMessage.AppendFormat(L"%x",
			debug_event.u.LoadDll.lpBaseOfDll);
		break;

	case UNLOAD_DLL_DEBUG_EVENT:
```

```
            stringEventMessage.Format(L"%s",
                  DllMap[debug_event.u.UnloadDll.lpBaseOfDll] );
            break;

      case OUTPUT_DEBUG_STRING_EVENT:
            OUTPUT_DEBUG_STRING_INFO & DebugString =
                  debug_event.u.DebugString;

            WCHAR *message=new WCHAR[DebugString.nDebugStringLength];

            ZeroMemory(message, DebugString.nDebugStringLength);

            ReadProcessMemory(processInfo.hProcess,
                  DebugString.lpDebugStringData, message,
                  DebugString.nDebugStringLength, NULL);

            if ( DebugString.fUnicode )
                  stringEventMessage = message;
            else
                  stringEventMessage = (LPSTR)message;

            delete []message;
            break;

      case EXCEPTION_DEBUG_EVENT:

            EXCEPTION_DEBUG_INFO& exception = debug_event.u.Exception;
            switch( exception.ExceptionRecord.ExceptionCode){
                  case STATUS_BREAKPOINT:
                        stringEventMessage= "Break point";
                        break;
                  default:
                        if(exception.dwFirstChance == 1){
                        stringEventMessage.Format(L"First chance
                        exception at %x, exception-code: 0x%08x",

                        exception.ExceptionRecord.ExceptionAddress,

                        exception.ExceptionRecord.ExceptionCode);}
                        else {
                        stringEventMessage.Format(L"Second chance
                        exception at %x, exception-code: 0x%08x",
                        exception.ExceptionRecord.ExceptionAddress,
                        exception.ExceptionRecord.ExceptionCode);
                        }
                        dwContinueStatus = DBG_EXCEPTION_NOT_HANDLED;
                  }
                  break;
            }
      }

      SendMessage(DEBUG_EVENT_MESSAGE, (WPARAM) &stringEventMessage,   '
                              debug_event.dwDebugEventCode);

      ContinueDebugEvent(debug_event.dwProcessId, debug_event.dwThreadId,
                        dwContinueStatus);

      dwContinueStatus = DBG_CONTINUE;
}}
```

Лістинг 2. Обробка автодоповнення дужок

```
public void ProcessAutoCompleteBrackets(String s)
{
   int sel = textEditor.ActiveTextAreaControl.Caret.Offset;
    if (IsAutoCompeleBracesSelected() == true)
    {
        switch (s)
        {
            case "(":
                textEditor.Document.Insert(sel, ")");
                ischanged = false;
                ismouseclickontexteditor = true;
                break;

            case "[":
                textEditor.Document.Insert(sel, "]");
                ismouseclickontexteditor = true;
                ischanged = false;
                break;

            case "{":
                textEditor.Document.Insert(sel, "}");
                ischanged = false;
                ismouseclickontexteditor = true;
                break;

            case "'":
                textEditor.Document.Insert(sel, "'");
                ischanged = false;
                ismouseclickontexteditor = true;
                break;

            case "\"":
                textEditor.Document.Insert(sel, "\"");
                ischanged = false;
                ismouseclickontexteditor = true;
                break;

            case "<":
                if (this.Text.Contains(".html") ||
                      this.Text.Contains(".xml"))
                {
                    textEditor.Document.Insert(sel, ">");
                    ischanged = false;
                    ismouseclickontexteditor = true;
                }
                break;
        }
    }}
```

Лістинг 3. Клас *DarkMenuRenderer*

```
  public class DarkMenuRenderer : ToolStripRenderer
{
    public static void DrawRoundedRectangle(Graphics g, int x, int y, int
              width, int height, int m_diameter, Color color)
    {
        using (Pen pen = new Pen(color))
        {
            var BaseRect = new RectangleF(x, y, width, height);
            var ArcRect = new RectangleF(BaseRect.Location, new
              SizeF(m_diameter, m_diameter));
            g.DrawArc(pen, ArcRect, 180, 90);
            g.DrawLine(pen, x + Convert.ToInt32(m_diameter / 2), y, x +
              width - Convert.ToInt32(m_diameter / 2), y);
            ArcRect.X = BaseRect.Right - m_diameter;
            g.DrawArc(pen, ArcRect, 270, 90);
            g.DrawLine(pen, x + width, y + Convert.ToInt32(m_diameter / 2),
              x + width, y + height - Convert.ToInt32(m_diameter / 2));
            ArcRect.Y = BaseRect.Bottom - m_diameter;
            g.DrawArc(pen, ArcRect, 0, 90);
            g.DrawLine(pen, x + Convert.ToInt32(m_diameter / 2), y + height,
              x + width - Convert.ToInt32(m_diameter / 2), y + height);
            ArcRect.X = BaseRect.Left;
            g.DrawArc(pen, ArcRect, 90, 90);
            g.DrawLine(pen, x, y + Convert.ToInt32(m_diameter / 2), x, y +
              height - Convert.ToInt32(m_diameter / 2));
        }
    }
    protected override void
              OnRenderMenuItemBackground(ToolStripItemRenderEventArgs e)
    {

        base.OnRenderMenuItemBackground(e);
        if (e.Item.Enabled)
        {
            if (e.Item.IsOnDropDown == false && e.Item.Selected)
            {
                var rect = new Rectangle(2, 1, e.Item.Width - 5,
                    e.Item.Height - 2);
                Brush b2 = new System.Drawing.Drawing2D.
                LinearGradientBrush(e.Item.ContentRectangle,
                Color.FromArgb(255, 255, 255, 180),
                Color.FromArgb(255, 255, 234, 130), 90);
                e.Graphics.FillRectangle(b2, rect);
                DrawRoundedRectangle(e.Graphics, rect.Left - 1, rect.Top -
                    1, rect.Width, rect.Height + 1, 4, Color.Orange);
                e.Item.ForeColor = Color.Black;
            }
            else if (e.Item.IsOnDropDown && e.Item.Selected)
            {
                var rect = new Rectangle(2, 1, e.Item.Width - 4,
                    e.Item.Height - 2);
                Brush b2 = new System.Drawing.Drawing2D.
                LinearGradientBrush(e.Item.ContentRectangle,
                Color.FromArgb(255, 255, 255, 180),
                Color.FromArgb(255, 255, 234, 130), 90);
                e.Graphics.FillRectangle(b2, rect);
                DrawRoundedRectangle(e.Graphics, rect.Left - 1, rect.Top -
                    1, rect.Width, rect.Height + 1, 4, Color.Orange);
                e.Item.ForeColor = Color.Black; }
            else
            {
                e.Item.ForeColor = Color.Black;
            }
```

```
            if ((e.Item as ToolStripMenuItem).DropDown.Visible &&
                    e.Item.IsOnDropDown == false){
                var rect = new Rectangle(2, 1, e.Item.Width - 4,
                          e.Item.Height - 2);
                var rect2 = new Rectangle(2, 1, e.Item.Width - 4,
                          e.Item.Height - 2);
                Brush b2 = new System.Drawing.Drawing2D.
                LinearGradientBrush(e.Item.ContentRectangle,
                Color.FromArgb(255, 255, 255, 180),
                Color.FromArgb(255, 255, 234, 130), 90);
                e.Graphics.FillRectangle(b2, rect);
                e.Graphics.DrawRectangle(new Pen(Brushes.Orange), rect2);
                e.Item.ForeColor = Color.Black;
            }}}
    protected override void
        OnRenderItemCheck(ToolStripItemImageRenderEventArgs e)
    {
        base.OnRenderItemCheck(e);

        if (e.Item.Selected) {
            var rect = new Rectangle(3, 1, 20, 20);
            var rect2 = new Rectangle(4, 2, 18, 18);
            SolidBrush b = new SolidBrush(Color.Orange);
            SolidBrush b2 = new SolidBrush(Color.FromArgb(255, 220, 230,
                                                          230));
            e.Graphics.FillRectangle(b, rect);
            e.Graphics.FillRectangle(b2, rect2);
            e.Graphics.DrawImage(e.Image, new Point(5, 3));}
        else {
            var rect = new Rectangle(3, 1, 20, 20);
            var rect2 = new Rectangle(4, 2, 18, 18);
            SolidBrush b = new SolidBrush(Color.Blue);
            SolidBrush b2 = new SolidBrush(Color.FromArgb(255, 240, 250,
                                                          250));
            e.Graphics.FillRectangle(b, rect);
            e.Graphics.FillRectangle(b2, rect2);
            e.Graphics.DrawImage(e.Image, new Point(5, 3));}}
    protected override void
        OnRenderSeparator(ToolStripSeparatorRenderEventArgs e){
        base.OnRenderSeparator(e);
        var DarkLine = new SolidBrush(Color.FromArgb(160, 160, 160));
        var WhiteLine = new SolidBrush(Color.FromArgb(160, 160, 160));
        var rect = new Rectangle(30, 3, e.Item.Width - 32, 1);
        e.Graphics.FillRectangle(DarkLine, rect);
        e.Graphics.FillRectangle(WhiteLine, rect);
    }
    protected override void OnRenderImageMargin(ToolStripRenderEventArgs e)
    {
        base.OnRenderImageMargin(e);

        var rect = new Rectangle(0, 0, e.ToolStrip.Width,
                    e.ToolStrip.Height);
        e.Graphics.FillRectangle(new SolidBrush(Color.FromArgb(255, 190,
                    210, 250)), rect);
        var DarkLine = new SolidBrush(Color.FromArgb(255, 240, 240, 250));
        var rect2 = new Rectangle(1, 2, 24, e.AffectedBounds.Height);
        e.Graphics.FillRectangle(DarkLine, rect2);

        var rect3 = new Rectangle(0, 0, e.ToolStrip.Width - 1,
                                  e.ToolStrip.Height - 1);
        e.Graphics.DrawRectangle(new Pen(Brushes.DarkGray), rect3);
    }
}
```

Лістинг 4. Форма пошуку з заміною

```
 public partial class Replace_Form : Form
{
    TextEditorControl texteditor;
    public Replace_Form(TextEditorControl tx)
    {
        InitializeComponent();
        texteditor = tx;
    }


    private void textBox1_TextChanged(object sender, EventArgs e)
    {
        if (textBox1.Text == "" || textBox2.Text == "")
        {
            ReplaceAllButton.Enabled = false;
        }
        else
        {
            ReplaceAllButton.Enabled = true;
        }
    }


    private void textBox2_TextChanged(object sender, EventArgs e)
    {
        if (textBox1.Text == "" || textBox2.Text == "")
        {
            ReplaceAllButton.Enabled = false;
        }
        else
        {
            ReplaceAllButton.Enabled = true;
        }
    }



    private void ReplaceAllButton_Click(object sender, EventArgs e)
    {
        String findtext = textBox1.Text;
        String replacetext = textBox2.Text;
        if (findtext != "" && replacetext != "")
        {
            RichTextBox rtb = new RichTextBox();
            rtb.Text = texteditor.Text;
            rtb.Text = rtb.Text.Replace(findtext, replacetext);
            texteditor.Text = rtb.Text;
        }
    }


    private void CloseButton_Click(object sender, EventArgs e)
    {
        this.Close();
    }


    private void Replace_Form_Load(object sender, EventArgs e)
    {
        ReplaceAllButton.Enabled = false;
    }
}
```

Лістинг 5. Обробка запиту на компіляцію

```
public void Process_CompileAsampl()
 {
     if (IsAutoCompileAsamplSelected() == true)
    {
         String filename = GetTabPageOfComponent(this).Text;
         filename = filename.TrimEnd(new Char[] { '*' });
         String asamplpath = getasamplpath();
         String fullpath = getCurrentProjectLocationFolder() +  "\\" + filename;
         Boolean ifExists = File.Exists(fullpath);

         if (ifExists)
         {
             try
             {
                 StreamWriter str;
                 System.IO.File.WriteAllText(getCurrentProjectLocationFolder() +
                    filename, "");
                 str = System.IO.File.AppendText(getCurrentProjectLocationFolder() +
                    filename);
                 str.Write(textEditor.Text);
                 str.Close();
             }
             catch { }

             process = new Process();
             this.CompileASAMPL(filename, asamplpath);
         }
     }
 }

 public void CompileASAMPL(String file, String compilerpath)
 {
     if (this.Compile(compilerpath + "\\Asampl.exe", Path.GetDirectoryName(file),
              Path.GetFileName(file)))
     {
         ErrorReader = process.StandardOutput;
         string response = ErrorReader.ReadToEnd();
         F.BuildLogTextBox.Text = response;
         if (response.Contains("expected"))
         {
             F.BuildLogTextBox.Text = response;
         }
         else if (response == "")
         {
             F.BuildLogTextBox.Text = "Program Compiled
                    Successfully.................!";
         }
         else
         {
             F.BuildLogTextBox.Text = "Program Compiled
                    Successfully.................!";
         }
     }
     else
     {
         F.BuildLogTextBox.Text = "Program Compiled Successfully.................!";
     }
 }
```

Лістинг 6. Обробка запиту на закриття проекту

```
private void File_CloseProjectMenuItem_Click(object sender, EventArgs e)

    {
        if (this.Text != "ASAMPL-IDE")
        {
            if (myTabControl.TabCount > 0)
            {
                System.Windows.Forms.TabControl.TabPageCollection tabcoll =
                         myTabControl.TabPages;

                foreach (TabPage tabpage in tabcoll)
                {
                    if (tabcoll.Count > 1)
                    {
                        myTabControl.SelectedTab = tabpage;
                    }

                    if (tabpage.Text.Contains("*"))
                    {

                        String tabtext = tabpage.Text;

                        DialogResult dg = MessageBox.Show("Do you want to
                          save modified file  " + tabtext + "  before close
                          ?", "Save or Not", MessageBoxButtons.YesNo);
                        if (dg == DialogResult.Yes)
                        {
                            File_SaveMenuItem_Click(sender, e);
                            this.Text = "ASAMPL-IDE";

                    myTabControl.TabPages.Remove(myTabControl.SelectedTab);
                            ProjectExplorerTreeView.Nodes.Clear();
                            SetVisibilityOfToolStripButtons();
                            UpdateWindowsList_WindowMenu();
                            myTabControl_SelectedIndexChanged(sender, e);
                            File.WriteAllText(Application.StartupPath +
                    "\\files\\files.asamplfile", "");
                            LineToolStripLabel.Text = "Line";
                            ColumnToolStripLabel.Text = "Col";
                        }
                        else
                        {
                            this.Text = "ASAMPL-IDE";

                    myTabControl.TabPages.Remove(myTabControl.SelectedTab);
                            ProjectExplorerTreeView.Nodes.Clear();
                            SetVisibilityOfToolStripButtons();
                            UpdateWindowsList_WindowMenu();
                            myTabControl_SelectedIndexChanged(sender, e);
                            File.WriteAllText(Application.StartupPath +
                                "\\files\\files.asamplfile", "");
                            LineToolStripLabel.Text = "Line";
                            ColumnToolStripLabel.Text = "Col";
                        }
                    }
                    else
                    {
                        isFilechanged = false;
                        this.Text = "ASAMPL-IDE";

                    myTabControl.TabPages.Remove(myTabControl.SelectedTab);
                        ProjectExplorerTreeView.Nodes.Clear();
                        SetVisibilityOfToolStripButtons();
                        UpdateWindowsList_WindowMenu();
```

```
                    myTabControl_SelectedIndexChanged(sender, e);
                    File.WriteAllText(Application.StartupPath +
                "\\files\\files.asamplfile", "");
                    LineToolStripLabel.Text = "Line";
                    ColumnToolStripLabel.Text = "Col";
                }
            }
        }
        else
        {
            this.Text = "ASAMPL-IDE";
            isFilechanged = false;
            ProjectExplorerTreeView.Nodes.Clear();
            FilenameToolStripLabel.Text = "ASAMPL-IDE";
            File.WriteAllText(Application.StartupPath +
          "\\files\\files.asamplfile", "");
            LineToolStripLabel.Text = "Line";
            ColumnToolStripLabel.Text = "Col";
            SetVisibilityOfToolStripButtons();
        }
    }
  }
```

Лістинг 7. Обробка автодоповнення дужок

```
public void ProcessCodeCompletionAction(String key) {
      if (IsAutoCompeletionModeSelected()){
          EnteredKey = "";
          EnteredKey = EnteredKey + key;

          this.GetKeywordsListFromFile();

          keywordslist.Sort();

          ToolTipControl.Visible = false;

          char ch;
          for (ch = 'a'; ch <= 'z'; ch++){
          if (key == ch.ToString()){
              CodeCompleteBox.Items.Clear();
              foreach (String item in keywordslist){
                  if (item.StartsWith(EnteredKey))
                          CodeCompleteBox.Items.Add(item);}
              foreach (String item in keywordslist){
                  if (item.StartsWith(EnteredKey)){
                      CodeCompleteBox.SelectedItem = item;
                      CodeCompleteBox.Cursor = Cursors.Default;
                   CodeCompleteBox.Size = new
                        System.Drawing.Size(this.getWidth(),
      this.getHeight() +

(int)textEditor.ActiveTextAreaControl.TextArea.Font.Size);
                  CodeCompleteBox.Location = this.getXYPoints();
                              textEditor.ActiveTextAreaControl.
                        TextArea.Controls.Add(CodeCompleteBox);
                  CodeCompleteBox.Focus();
                  isCodeCompleteBoxAdded = true;

                     ToolTipControl.Location = new
                       Point(CodeCompleteBox.Location.X +
            CodeCompleteBox.Width,
                       CodeCompleteBox.Location.Y);
                     ToolTipControl.Visible = true;
```

```
            this.ProcessToolTips(CodeCompleteBox.SelectedItem.ToString());


                  textEditor.ActiveTextAreaControl.
                  TextArea.Controls.Add(ToolTipControl);

                     isToolTipControlAdded = true;
                  break;}
                   else
                          isCodeCompleteBoxAdded = false;
             }
             else if (key == ch.ToString().ToUpper()){
                  CodeCompleteBox.Items.Clear();
                  foreach (String item in keywordslist){
                      if (item.StartsWith(EnteredKey))
                          CodeCompleteBox.Items.Add(item);}
                  foreach (String item in keywordslist){
                      if (item.StartsWith(EnteredKey))
                      {
                          CodeCompleteBox.SelectedItem = item;
                          CodeCompleteBox.Cursor = Cursors.Default;
                          CodeCompleteBox.Size = new

                        System.Drawing.Size(this.getWidth(),
            this.getHeight() +

            (int)textEditor.ActiveTextAreaControl.TextArea.Font.Size);
                          CodeCompleteBox.Location = this.getXYPoints();

                          textEditor.ActiveTextAreaControl.
                        TextArea.Controls.Add(CodeCompleteBox);
                          CodeCompleteBox.Focus();
                          isCodeCompleteBoxAdded = true;

                          ToolTipControl.Location = new

                        Point(CodeCompleteBox.Location.X +
                        CodeCompleteBox.Width,
                        CodeCompleteBox.Location.Y);
                              this.ProcessToolTips(CodeCompleteBox.
                        electedItem.ToString());
                          textEditor.ActiveTextAreaControl.
                        TextArea.Controls.Add(ToolTipControl);

                              isToolTipControlAdded = true;

                          break;
                      }

                      else
                      {
                          isCodeCompleteBoxAdded = false;
                      }
                  }
              }
          }
      }
  }
```

Додаток 3
Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

СЕРЕДОВИЩЕ РОЗРОБЛЕННЯ ПРОГРАМ НА МОВІ ASAMPL

Виконав: Крисюк Андрій Михайлович

Керівник: доцент кафедри ПЗКС, к.т.н., доцент Сулема Євгенія Станіславівна

Київ – 2019

ПОСТАНОВКА ЗАДАЧІ

Мета проекту: розробити середовище розроблення та відлагоджувач програм для мови ASAMPL.

Завдання:

1. Проаналізувати архітектуру та типові інструменти, реалізовані в існуючих рішеннях.
2. Розробити архітектуру програмного застосунку.
3. Протестувати реалізовані функціональні можливості.

АКТУАЛЬНІСТЬ

На сьогодні не існує ані середовища розроблення програм на мові ASAMPL, ані засобів відлагоджування для них.

Створення подібного програмного застосунку для мови ASAMPL сприятиме спрощенню роботи з нею.

ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

- Простота синтаксису;
- Наявність чітко визначених стандартів.
- Наявність вичерпної документації, сторонніх якісних бібліотек;
- Наявність потужної бібліотеки для розроблення графічних інтерфейсів, та редактору для зручного їх створення.
- Статична типізація;
- Можливість управління пам’яттю.

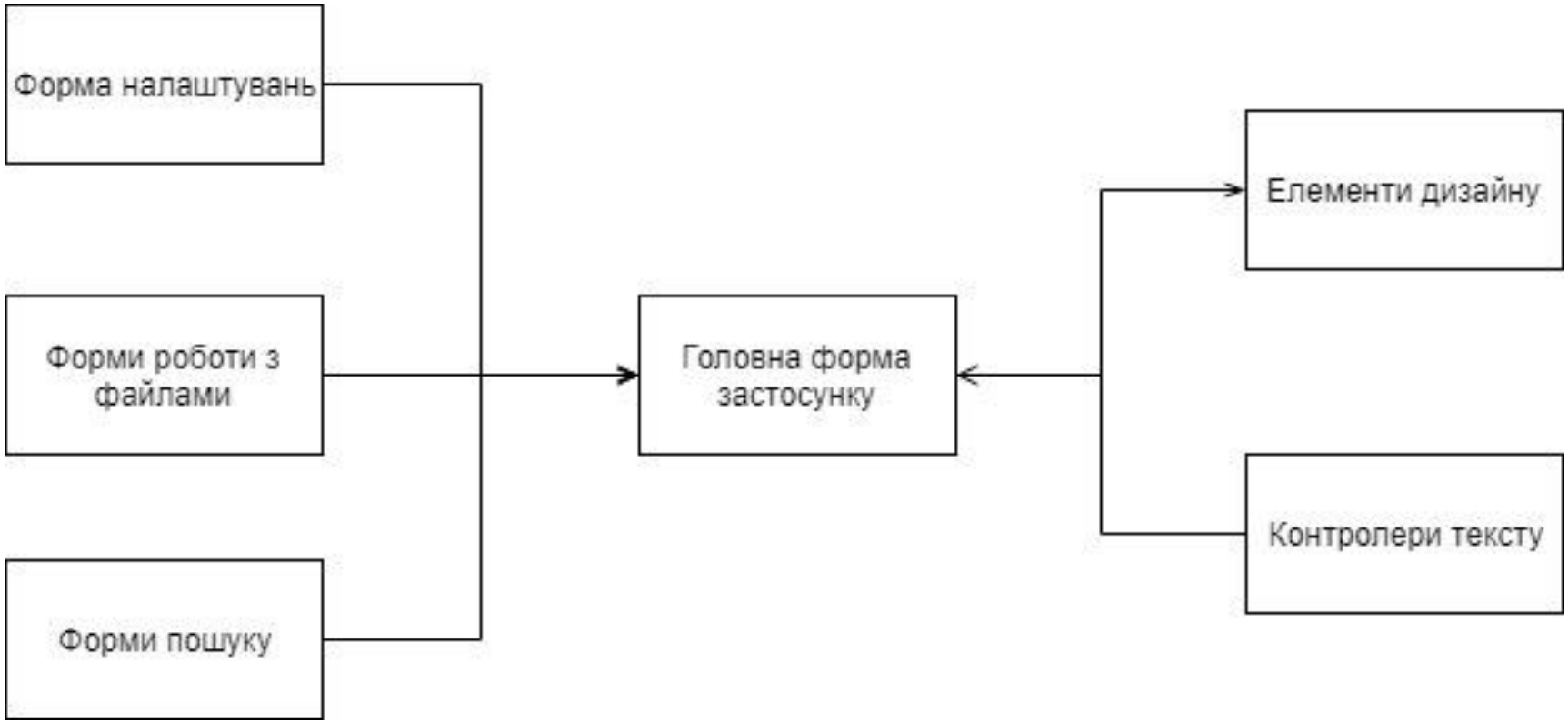
ВИБІР МОВИ ПРОГРАМУВАННЯ

Мова програм. / Критерії	C++	C#	Java
Наявність особливостей			
Стандарти	Так	Так	Так
Статична типізація	Так	Так	Так
Можливість управління пам'яттю	Так	Ні	Ні
Експертна оцінка			
Простота синтаксису	4	5	5
Вичерпність довідкової документації	4	5	4
Зручність використання графічних бібліотек	5	5	4
Зручність використання редакторів графічних інтерфейсів	5	5	4
Середня оцінка	**4,5**	**5**	**4,25**

КОМПОНЕНТИ СИСТЕМИ

1. Середовище розроблення:
   - форми
   - елементи форм
2. Відлагоджувач.

АРХІТЕКТУРА СЕРЕДОВИЩА РОЗРОБЛЕННЯ

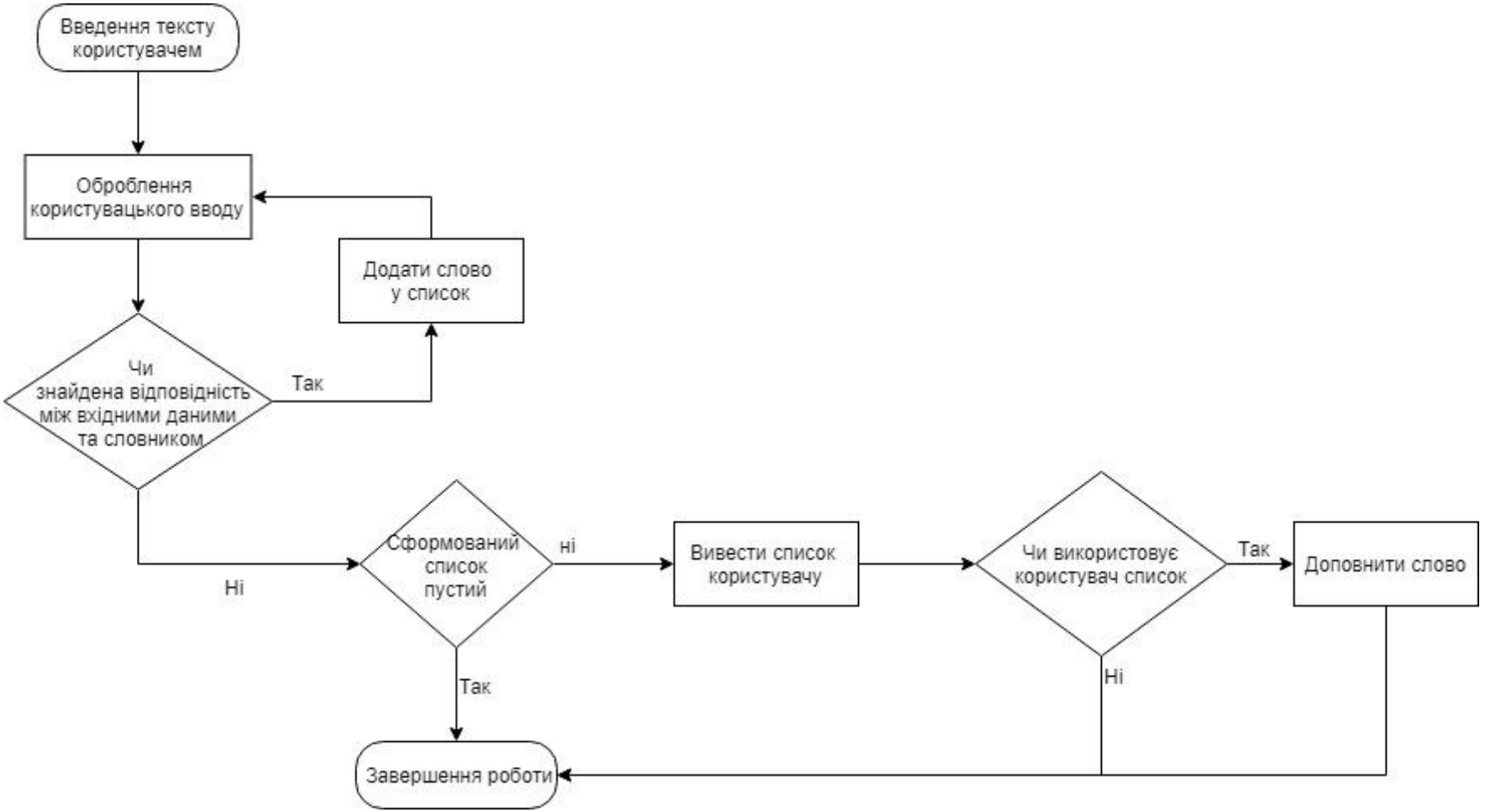


ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ СЕРЕДОВИЩА РОЗРОБКИ

- Створення проектів та файлів з кодом;
- Текстовий редактор з типовими функціями;
- Автодоповнення коду;
- Підсвітка синтаксису;
- Компіляція та запуск коду всередині середовища;
- Запуск відлагоджувача.

АЛГОРИТМ СИНТАКСИЧНОГО АНАЛІЗУ

АЛГОРИТМ АВТОДОПОВНЕННЯ

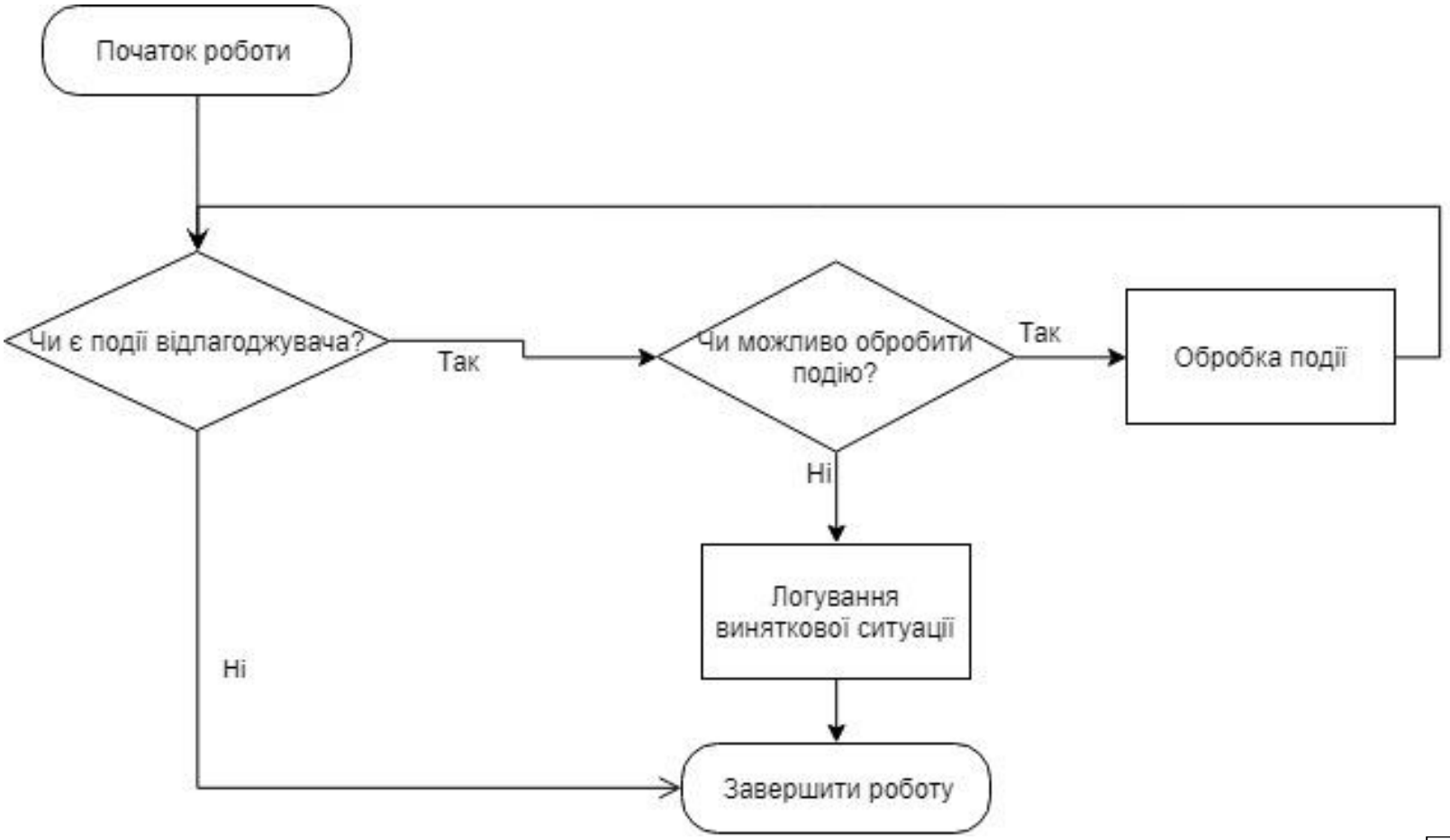


ВІДЛАГОДЖУВАЧ

Відлагоджувач дозволяє отримати інформацію про:

- Запуск та завершення нових процесів.
- Запуск та завершення нових потоків.
- Завантаження та вивантаження динамічних бібліотек.
- Виникнення виняткових ситуацій.

АЛГОРИТМ РОБОТИ ВІДЛАГОДЖУВАЧА

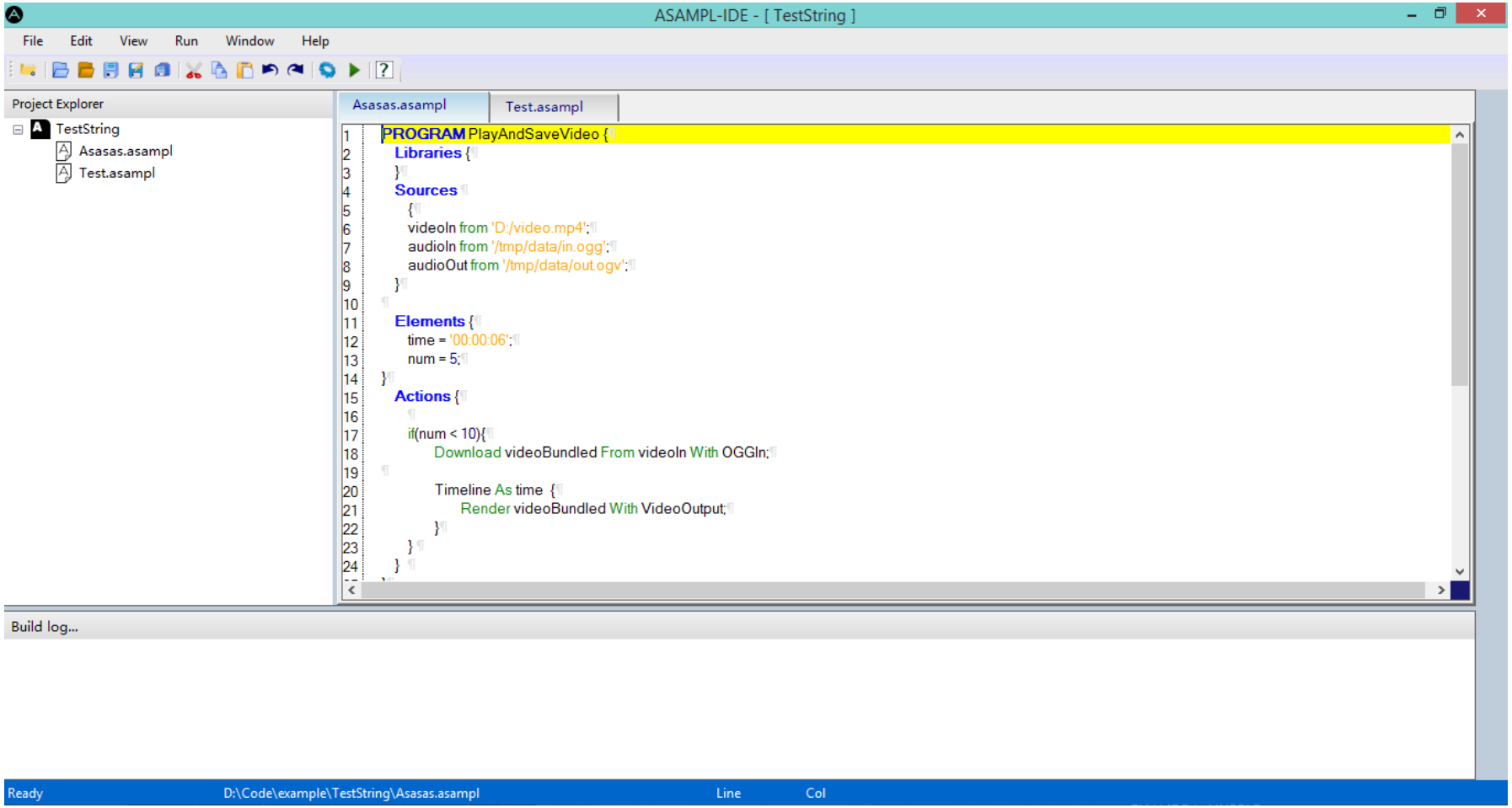


ПОРІВНЯННЯ СИСТЕМИ З АНАЛОГАМИ

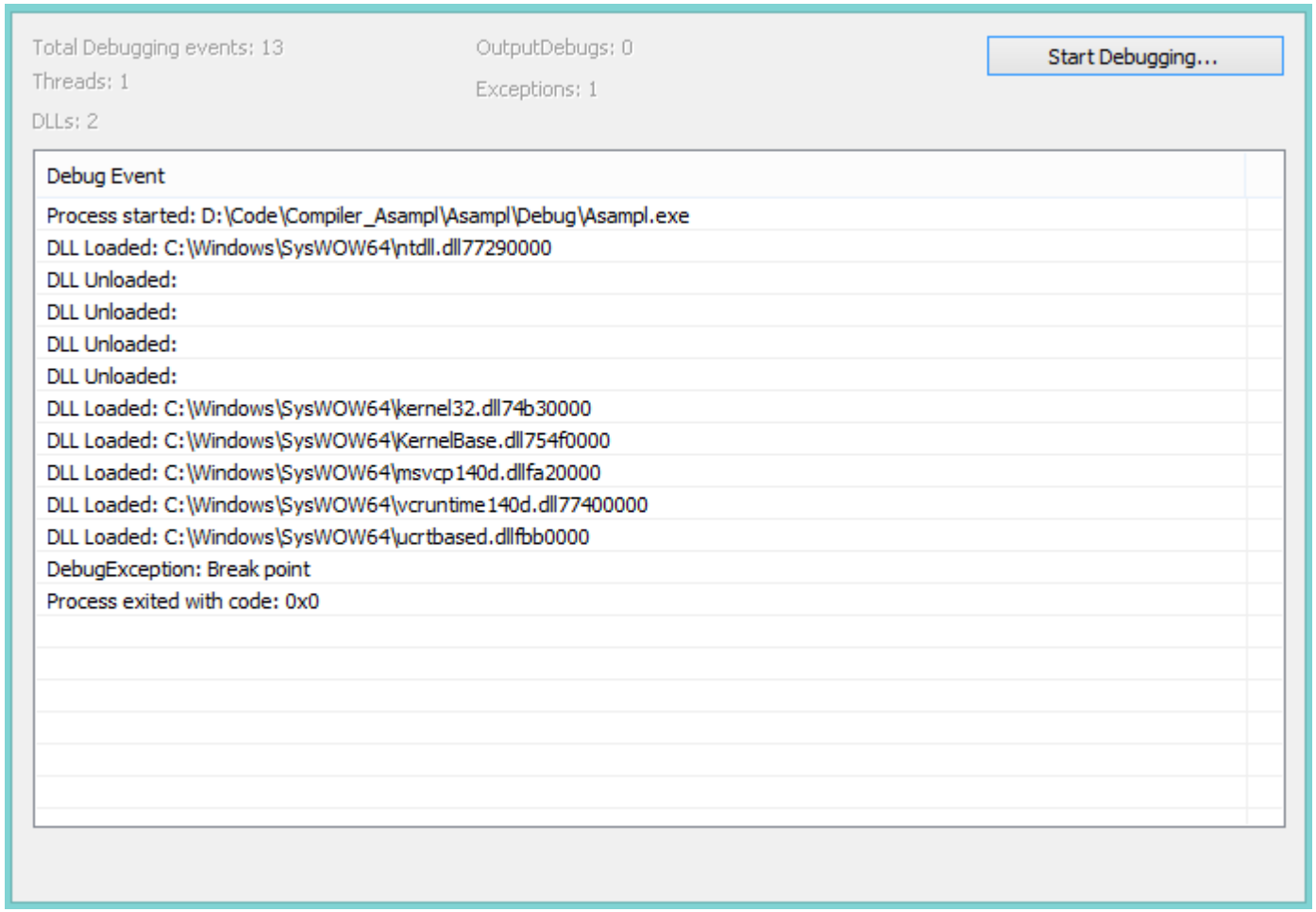
Функція	**ASAMPL IDE**	**Visual Studio**	**Eclipse**
Редактор коду	Так	Так	Так
Автодоповнення	Так	Так	Так
Підсвічення коду	Так	Так	Так
Перевірка коду перед компіляцією	Ні*	Так	Так
Запуск програмного коду	Так	Так	Так
Відлагоджувач	Так	Так	Так

*Для перевірки коду перед запуском можливо використати функцію автокомпіляції

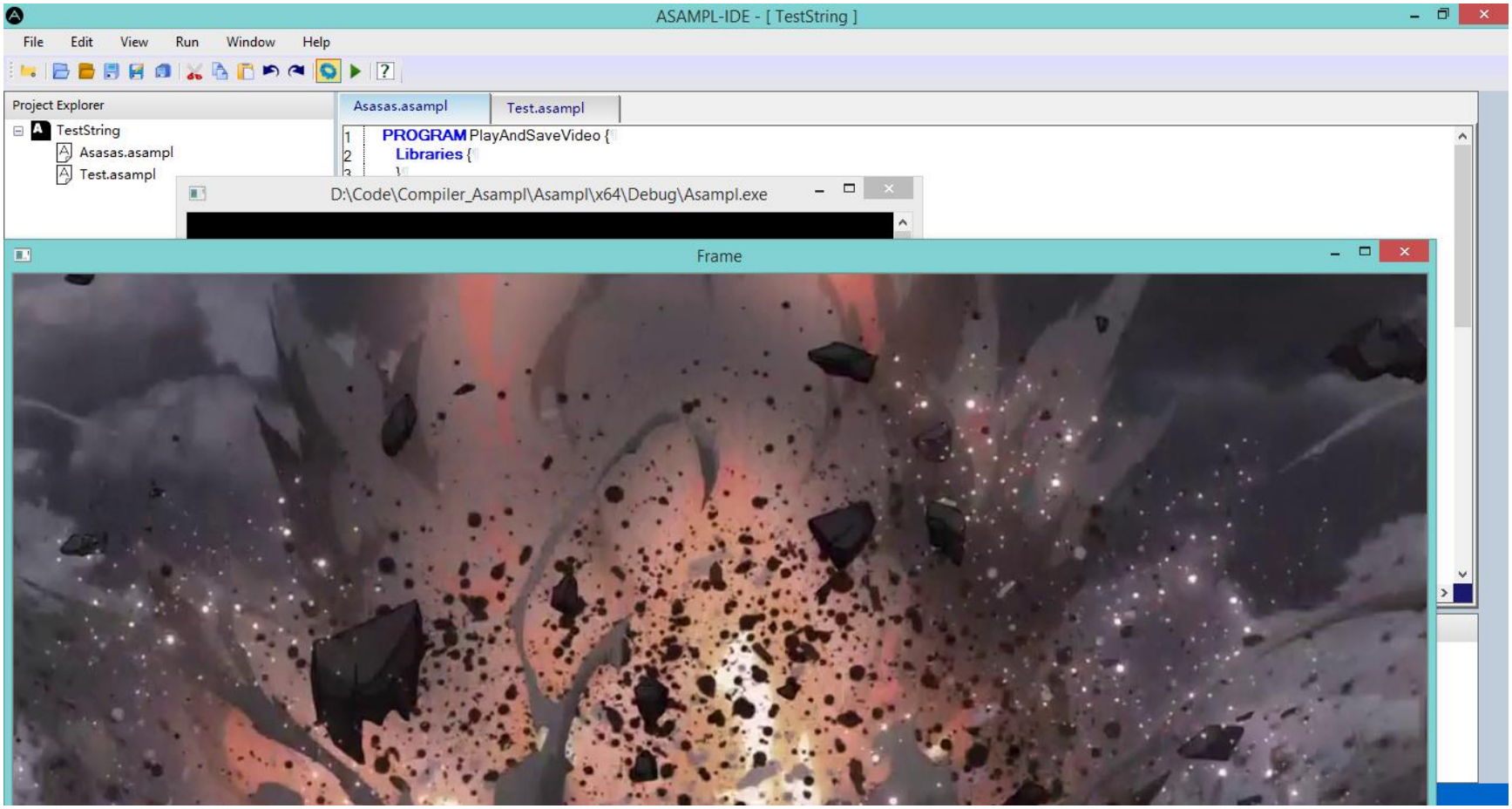
ПРИКЛАД РОБОТИ ПРОГРАМИ



ПРИКЛАД РОБОТИ ПРОГРАМИ



ПРИКЛАД РОБОТИ ПРОГРАМИ



ВИСНОВКИ

1. Створено середовище розроблення програм на мові ASAMPL.
2. Проаналізовано існуючі аналоги для виявлення особливостей архітектури та функціональних можливостей.
3. Розроблено та реалізовано архітектуру системи та основні функціональні можливості, заявлені в технічному завданні.
4. Створену систему було протестовано на працездатність та відповідність вимогам технічного завадання.

УНІКАЛЬНІСТЬ

Розділ	Процент унікальності, %
Вступ	98
Перший розділ	93
Другий розділ	96
Третій розділ	92
Четвертий розділ	99
Висновок	91
Весь текст	92

Дякую за увагу!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

СЕРЕДОВИЩЕ РОЗРОБЛЕННЯ ПРОГРАМ НА МОВІ ASAMPL

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проекту:

_____ Є.С. Сулема

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ А.М. Крисюк

2018

ЗМІСТ

1. Об’єкт випробувань 3
2. Мета тестування 3
3. Методи тестування 3
4. Засоби та порядок тестування 4

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Середовище розроблення програм на мові ASAMPL, який являє собою застосунок для ОС Windows, частина якого написана мовою C# за допомогою бібліотеки Windows Forms, а інша частина за допомогою мови C++.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) функціональна працездатність елементів графічного інтерфейсу;
2) забезпечення роботи з файлами та проектами;
3) коректність роботи функцій синтаксичного аналізатора та автодоповнення коду;
4) забезпечення взаємодії середовища розроблення та компілятора мови ASAMPL;
5) зручність роботи із системою;
6) відповідність вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. На відповідність функціональним вимогам перевіряються як окремі компоненти, так і безпосередньо програмний продукт загалом.

Використовуються наступні методи:

1) функціональне тестування, зокрема на рівні Critical path – базове тестування;
2) тестування продуктивності програмного забезпечення, зокрема Stability testing (тестування стабільності);
3) димове тестування;
4) тестування інтерфейсу.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Працездатність застосунку перевіряється з використанням:

1) динамічного ручного тестування на відповідність функціональним вимогам;
2) статичного тестування коду;
3) покриття юніт-тестами основних функцій з використанням NUnit та .NET Core.
4) тестування зручності використання;
5) тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

СЕРЕДОВИЩЕ РОЗРОБЛЕННЯ ПРОГРАМ НА МОВІ ASAMPL

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проекту:

_____ Є.С. Сулема

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ А.М. Крисюк

2019

ЗМІСТ

1. Опис структури застосунку 3
2. Опис форми налаштуваннь 5
3. Робота з файлами та проектами 6
4. Робота з редактором коду 7
5. Компіляція коду 8
6. Відлагодження коду 9

1. Опис структури застосунку

Головне вікно програми (рис. 1) складається з шести елементів:

- панелі меню;
- панелі інструментів;
- дерева файлів проекту;
- редактору коду;
- логу компіляції;
- панелі статусу.

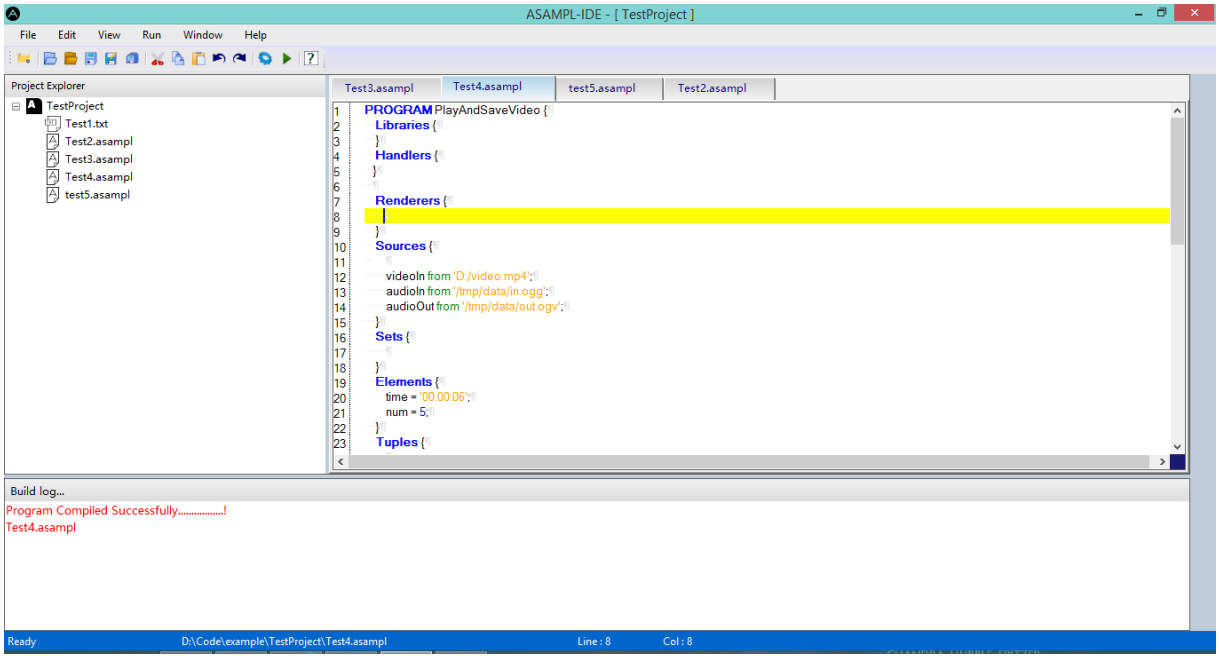


Рис. 1. Головне вікно програми

Панель меню складається з шести меню. Меню *File* надає користувачу доступ до створення окремих файлів та проектів, їх збереження, закриття, видалення, а також дозволяє відкрити останній завантажений проект без пошуку його у файловій системі.

Меню *Edit* представляє інтерфейс для роботи з файлами вихідного коду.

Меню *View* дозволяє користувачеві налаштовувати вигляд середовища розроблення.

Меню *Run* надає доступ до функцій компіляції, запуску та відлагодження програм.

Меню *Window* дозволяє швидко перезапустити середовище розроблення, закрити всі вкладки з кодом та перейти до кожної з відкритих вкладок.

Меню *Help* надає довідкові дані про роботу системи.

Наступним елементом інтерфейсу є панель інструментів, що надає швидкий доступ до часто використовуваних функцій, таких як створення нового проекту, відкриття існуючого проекту та файлів, зберігання вибраного файлу, зберігання зі зміною назви, зберігання всіх файлів, копіювання, вирізання та вставка тексту, відміна та повтор останньої дії, компіляція та запуск поточного файлу, доступ до довідкової інформації.

Дерево файлів проекту зберігає та виводить користувачу всі файли, що вказані у файлі проекту. Користувач має можливість відкривати ці файлі в редакторі коду. При натисканні правою кнопкою миші на дереві проекту, виводиться контекстне меню, що дозволяє створити нові файли та додати їх до проекту, відкрити вказаний файл у редакторі чи з використанням редактору встановленого системою, видалити файл та закрити проект.

Лог компіляції виводить інформацію про перебіг компіляції коду. Користувач може отримати повідомлення про успішний результат роботи компілятора, або повідомлення з помилкою, яку видає компілятор при обробці даного сегменту коду.

Панель статусу внизу головного вікна програми відображає інформацію про поточний відкритий проект та координати каретки в редакторі коду.

Основним елементом середовища розроблення є редактор коду. Панель вкладок у верхній частині редактору (чи нижній, якщо це вибрано в налаштуваннях системи) дозволяє користувачу переходити між відкритими у редакторі файлами. При натисканні правою кнопкою миші на

назві вкладки відкривається контекстне меню, що дозволяє зберегти вибраний файл, закрити його, закрити всі вкладки (з можливістю залишити обрану вкладку відкритою). Також користувач може скопіювати повний шлях файлу у файловій системі до буферу або відкрити директорію, у якій знаходиться вибраний файл.

2. Опис форми налаштувань

Форма налаштувань складається з двох вкладок, що відповідають за різні налаштування системи.

Перша вкладка (рис. 2) містить налаштування всього середовище розроблення, а саме кольорову схему, шлях до компілятора, наявність та розміщення елементів графічного інтерфейсу.

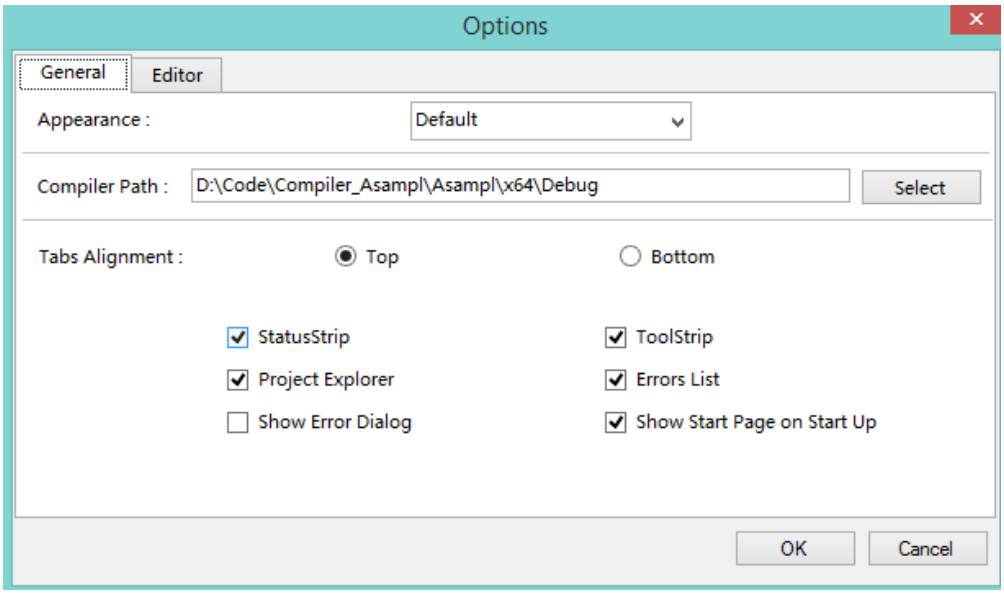


Рис. 2. Вкладка налаштувань системи

Друга вкладка (рис. 3) містить налаштування, що стосуються редактору коду: шрифт та його розмір, наявність номерів рядків коду, виділення помилок, автодоповнення дужок та пошук пар дужок у коді та інше.

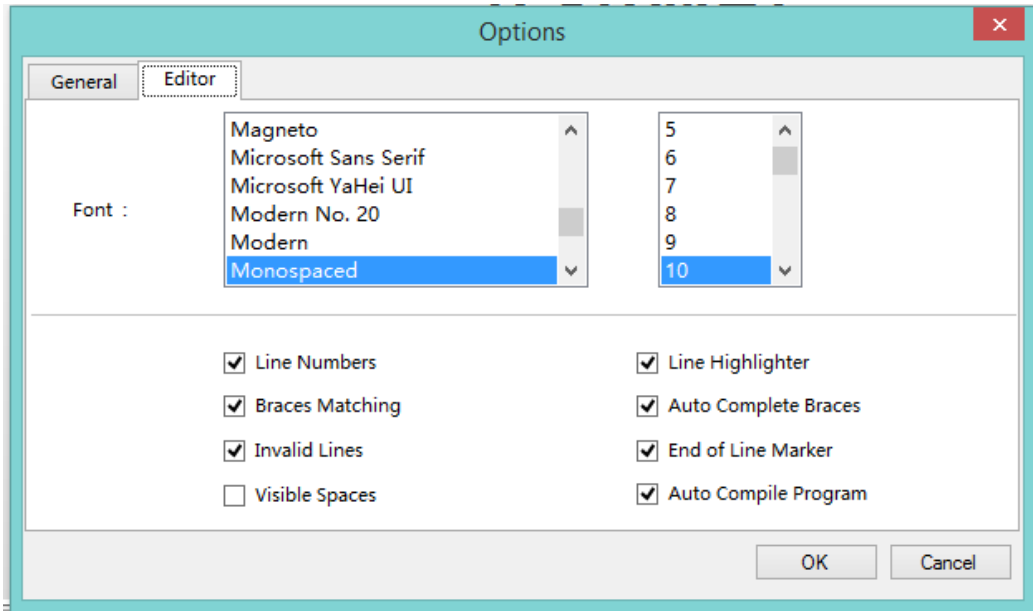


Рис. 3. Вкладка налаштувань редактору

3. Робота з файлами та проектами

Користувач може створювати нові файли та проекти декількома шляхами. Перший – використання меню File та його підменю New. При створенні нового проекту користувачу відкривається форма (рис. 4), що дозволяє визначити шлях до директорії проекту та його назву. Аналогічні форми відриваються при створенні файлів.

Окрім використання меню File, створити проект можна за допомогою відповідної кнопки панелі інструментів чи комбінації гарячих клавіш CTRL+ALT+N.

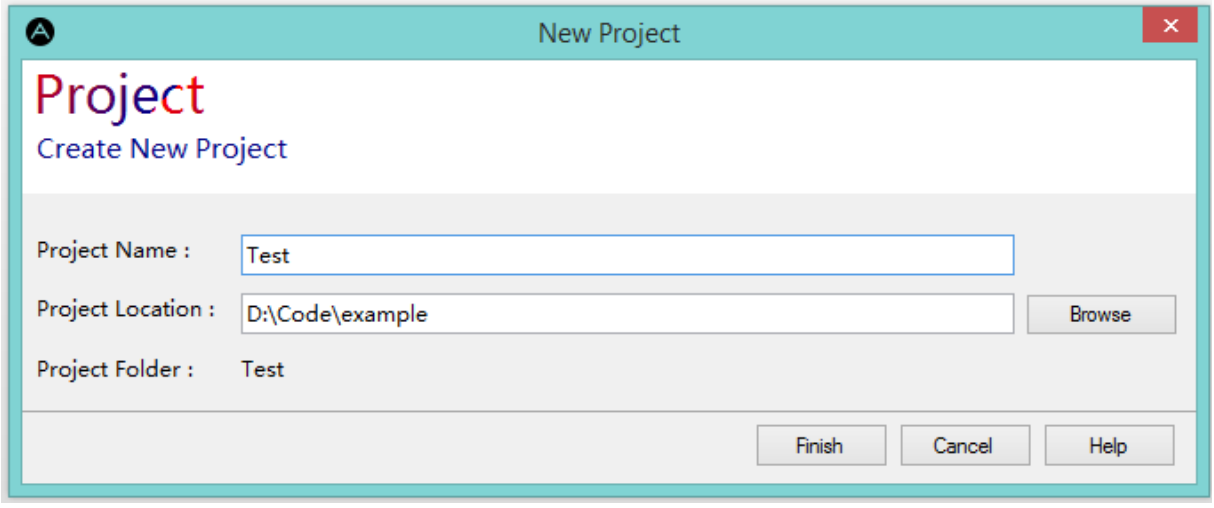


Рис. 4. Форма створення нового проекту

Альтернативним шляхом створення файлів є використання контекстного меню панелі дерева файлів проекту. Для цього необхідно натиснути правою клавішею миші на панелі та вибрати необхідний пункт із меню.

Для відкриття проектів також можна використати меню View, панель інструментів та комбінацію клавіш CTRL+SHIFT+O. Окрім цього в меню View є доступ до останнього відкритого проекту, а функції відкриття файлів, відкриття та створення проектів доступні зі стартової сторінки редактору.

4. Робота з редактором коду

Редактор коду, окрім стандартних функцій роботи з текстом, також надає можливість синтаксичного аналізу та автодоповнення коду. Текстові функції, наприклад, вирізання, вставка, повтор останніх дій, працюють аналогічно будь-якому текстовому редактору з використанням відповідних комбінацій гарячих клавіш.

Синтаксичний аналіз коду виділяє синім кольором та жирним ключові слова мови, що описують структури програми, зеленим кольором та звичайним шрифтом оператори, помаранчевим кольором строки в лапках, та зеленим кольором коментарі. Приклад роботи синтаксичного аналізу можна побачити на рис. 1.

Автодоповнення представляє три функції:

- доповнення дужок;
- доповнення ключових слів;
- вставка шаблонів коду.

Доповнення дужок можна відключити в налаштуваннях редактору коду. Доповнення ключових слів показує користувачу вікно з ключовими словами, що починаються з символів, що введено в редакторі. Для

закінчення слова необхідно натиснути клавішу Enter. Навігація у списку слів відбувається за допомогою клавіш стрілок.

Вставка шаблонів працює з деякими ключовими словами, та доповнює умовні, циклічні конструкції відповідними символами. Наприклад, якщо користувач введе слово "if" та натисне клавішу F2, програма автоматично доповнить це слово синтаксичною структурою, що буде включати умову в круглих дужках та тіло умовної конструкції в фігурних дужках.

5. Компіляція коду

Запустити компілятор можна лише за умови відкритого проекту та файлу у ньому. Для запуску користувач може використати як панель інструментів, так і комбінацію гарячих клавіш ALT+C. Компілятор може прочитати як звичайний текстовий файл, так і файл коду ASAMPL. Особливістю поточної версії компілятору є те, що він не створює виконуваних файлів, тому код після компіляції одразу виконується. Якщо компіляція пройшла невдало, до логу компіляції виводиться повідомлення із вказаним номером рядку коду, де компілятор виявив помилку. В іншому випадку до логу буде виведено повідомлення про успішну компіляцію та виконано програму.

6. Відлагодження коду

Для запуску відлагоджувача коду необхідно натиснути відповідну кнопку панелі інструментів або запустити його через меню Run. Поточна версія відлагоджувача надає лише інформацію про перебіг роботи компілятора та програми в ньому, у вигляді подій завантаження динамічних бібліотек, створення нових потоків чи процесів, виникнення виняткових ситуацій. На рис. 5 наведено результат роботи відлагоджувача.

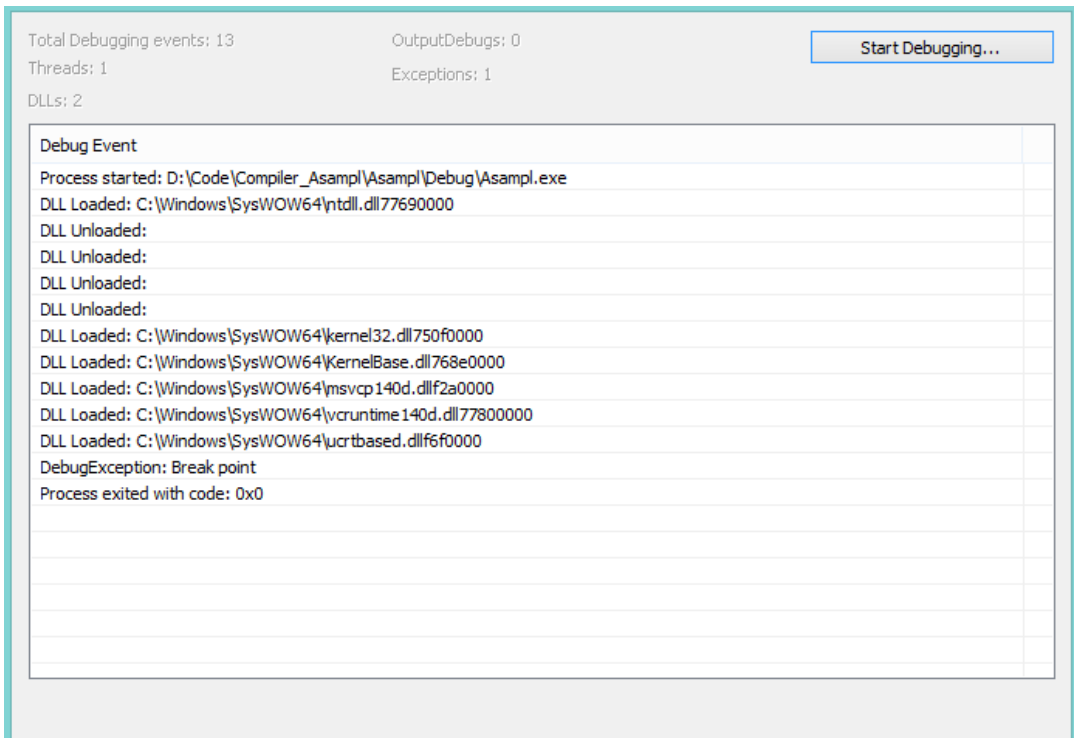


Рис. 5. Приклад роботи відлагоджувача